Requirements engineering for business workflow systems: a scenario-based approach

Strassl, Johann Gerhard

Requirements Engineering for Business Workflow Systems: A Scenario-based Approach

Johann Gerhard Strassl

Submitted for the degree of Doctor of Philosophy

University of Durham

Department of Computer Science

April 2001

Abstract

Workflow implementations require a deep understanding of business and human cooperation. Several approaches have been proposed to address this need for understanding, but largely in a descriptive way. Attempts to use them in software development have had mixed results.

The work reported here proposes that these approaches can be used in a generative way, as part of the requirement engineering process, by (a) extending requirements engineering modelling techniques with underlying cooperation properties, (b) integrating these techniques through the use of a derivation modelling approach, and (c) providing pragmatic heuristics and guidelines that support the real-world requirements engineering practitioner to ensure a high probability of success for the business workflow system to be developed.

This thesis develops and evaluates a derivation modelling approach that is based on scenario modelling. It supports clear and structured views of cooperation properties, and allows the derivation of articulation protocols from business workflow models in a scenario-driven manner. This enables requirements engineering to define how the expectations of the cooperative situation are to be fulfilled by the system to be built – a statement of requirements for business workflow systems that reflects the richness of these systems, but also acts as a feasible starting point for development.

The work is evaluated through a real-world case study of business workflow management.

The main contribution of this work is a demonstration that the above problems in modelling requirements for business workflow systems can be addressed by scenario-based derivation modelling approach. The method transforms models through a series of properties involving cooperation, which can be addressed by using what are effectively extensions of current requirements engineering methods.

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

I would like to express my sincere thanks to my supervisor, Dr. Cornelia Boldyreff, who made it possible for me to come to Durham and her help and guidance in the development of this thesis.

I would like to express my deep thanks to Dr. Simon Smith. Working together with him from the first day in Durham was very inspiring. His meticulous commenting on reports, ideas, and drafts is highly acknowledged.

I also would like to acknowledge my colleagues at Durham on the Toll-Bridge project, Phil Bevan and Gill Mallalieu, with whom I learnt a great deal during the early stages of this work.

I also thank my colleagues earlier at Softlab and now at Nortel Networks for interesting discussions.

I would like to thank those who gave me encouragement and support through other means, while I completed this work, in particular Prof. Keith Bennett and my family.

Finally, I would like to thank Sandra. Her love, encouragement, and support enabled me to finish this work.

I dedicate this thesis to my parents.

Declaration

This work was partly carried out between October 1994 and February 1997 while the author was working as a Research Assistant in the Department of Computer Science at the University of Durham on the Toll-Bridge project. This project was funded by ICI plc under the Strategic Research Fund. This earlier work appears in [Strassl 1996] and [Strassl and Smith 1997].

An early version of some material from Chapters 3 and 4 is presented in [Strassl and Smith 1998].

The complaint management case study presented in Chapter 5 is the original work of the author, which was carried out while working at Softlab and Nortel Networks in Munich, as all other work in this thesis, except as acknowledged in the text.

Statement of Copyright

1 Introduction

This chapter introduces software development for business workflow systems as a modelling process, and describes the role of requirements engineering within this process. An overview of the thesis is given, along with a synopsis of each of the following chapters.

1.1 Software development for business workflow systems

The development of software-based business workflow systems has been of increasing interest in both research and industry as the potential benefits of building systems which more closely support human processes become established.

Traditional software development has created many kinds of software products, and a diverse set of development methods associated with their production. However, software systems development is based on the understanding of a general

development process. In this general process, several activities must be carried out in order to develop the software system, and the products of one activity feed into, or back to, another. A typical software system development process consists of requirements engineering, design, implementation and maintenance (e.g. [Boehm 1988] [Sommerville 1992]). The final product should be a satisfactory software system.

A successful business workflow system supports the coordination and cooperation of work activities as is needed to satisfy business workflow processes.

This thesis is concerned with two aspects of software development for business workflow systems: requirements engineering and modelling. Each of these aspects is discussed briefly in the following sections, by way of introduction to the remainder of the thesis.

1.1.1 Requirements engineering

The core measure of the success of a software system is the degree to which it meets the objectives for which it was intended. In this sense, software systems requirements engineering is the process of discovering these objectives, by identifying stakeholders and their needs, and documenting these in an appropriate form that is amenable to analysis, communication, and subsequent implementation. However, it is well documented that there are a number of inherent difficulties in this process. Stakeholders (e.g.

customers, users, support staff, developers) may be numerous and distributed. They may not be involved in the process early enough. Their needs may vary and conflict, depending on the environment in which they work and the tasks they have to accomplish in the business. Their needs may not be to explicit or may be difficult to articulate and, inevitably, may be misidentified, which is one of the most significant sources of dissatisfaction with delivered software systems (e.g. [Macaulay 1996] [Lubars et al. 1993] [McGraw and Harbison 1997]).

The role of requirements engineering for software systems development is concerned "with real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families." [Zave 1997] The definition points out two main issues: first, it is real-world oriented since it highlights the importance of real-world goals that motivate the development of a software system. And second, the definition captures the evolution over time, which considers the reality of a changing world.

The context in which requirements engineering takes place is usually a human activity system, and the problem owners are people [Macaulay 1993]. Engagement in a requirements engineering process assumes that a new software system may be useful, but such a system will change the activities that it supports. Therefore, requirements engineering for business workflow systems needs to be sensitive to how people perceive and understand business

processes, how actors interact and cooperate in the business, and how it affects their work.

1.1.2 Modelling

Modelling is one of the fundamental activities in requirements engineering. It is the construction of abstract descriptions that are amenable to interpretation. For example, given a business process or work situation, a model represents abstract descriptions of the business processes in which the envisioned workflow system will operate.

Business workflow models are confronted with the aspects of cooperation and coordination. These aspects do not have an objective on its own but are prerequisites for the business to reach its objectives. As Bannon and Schmidt [Bannon and Schmidt 1991] say: "Cooperative work is constituted by work processes that are related as to content, that is, processes pertaining to the production of a particular product or service." Such model representations bring to light a variety of new problems, as it corresponds to much more fluid, less structured concepts and practices.

A variety of modelling approaches in requirements engineering exist (e.g. enterprise modelling, domain modelling, data modelling, goal-based approaches), which do not address such cooperation aspects sufficiently.

In this thesis, a major concern is the better understanding of the cooperative work aspects and how these can be used as part of the requirements engineering process for business workflow systems.

1.2 Problem statement

The overall problem addressed by this thesis is the activity of modelling and analysing requirements for business workflow systems. In general, there is a lack of support by current requirements engineering methods with respect to the provision of how the complexity of business work is managed.

Several approaches in sociology have been proposed to address the need for understanding cooperative work in a largely descriptive and analytical manner. Attempts to use these descriptive results in software development have had mixed results.

The ultimate goal of building a business workflow system is to solve some problem of a business (such as increase of profit or customer satisfaction). Understanding and analysing requirements of business processes results in a requirements specification, which is used for communication among stakeholders and may be part of a formal contract. Therefore, a requirements engineering method needs to describe workflow, actor behaviour and cooperation dependencies in an expressive manner that allows communication among people.

In Chapters 2 and 3, it is argued that a method to support the understanding of business and human cooperation as part of the requirements engineering process must work in a generative way that helps people think and explain complex behaviour. Thus, it must be possible:

- to extend requirements engineering modelling techniques with underlying cooperation properties,
- to integrate these techniques in such a way that it is clear how these can lead into and support each other, and
- to provide pragmatic heuristics and guidelines that support the real-world requirements engineering practitioner to ensure a high probability of success for the business workflow system to be built.

1.3 Assumptions

The main assumption of this thesis is that modelling and analysing requirements for business workflow systems should be visual scenario-based and of lightweight nature, in order to stimulate thinking and discussion about business workflow issues.

This is a reasonable, reality-proven assumption to make. It is adopted as an assumption rather than as a point to be demonstrated in greater length in this thesis, since the benefits of visual scenario-

based requirements elicitation and modelling have been convincingly demonstrated not just in principle (e.g. [Haumer et al. 1999]), but also increasingly in practice (e.g. [Weidenhaupt et al. 1998]).

1.4 Proposed solution

It has been shown that 'separation of concern' [Parnas 1972] approaches can lead to more manageable and traceable methods for analysis and reasoning. This thesis attempts to extend this result to the area of requirements engineering.

Particularly, this work suggests that a new approach, called *derivation modelling*, will:

- provide clear and structured views of cooperation properties,
- allow the derivation of articulation protocols from business workflow models in a scenario-driven manner,

and so provide a requirements engineering method that defines how the expectations of the cooperative situation are to be fulfilled by the workflow systems to be developed.

1.5 Method

The method chosen for developing and evaluating the proposed solution is as follows:

- identify characteristics which should form part of a derivation modelling method to model and analyse requirements for business workflow systems,
- devise a way of describing business workflow behaviour with this method,
- derive a set of heuristics to provide support for the modelling process,
- evaluate the work in the context of a real-world case study, and
- adjust the method according to the results of the case study.

The main decision made in the choice of this method was the possibility of using existing visual notations for investigating the derivation modelling approach. An alternative would have been to invent a new notation. However, this is not the objective of this thesis.

1.6 Results

This work reports the following main results:

- the identification of a set of underlying modelling properties for cooperation as part of the requirements engineering process,

- the development of a derivation modelling method based on scenarios, and

- the provision of pragmatic heuristics and guidelines that support requirement engineering practitioners.

1.7 Contribution

The contribution made by this thesis may be summarised as follows:

- An evaluation of a derivation modelling approach is given on the basis of scenarios for modelling requirements for business workflow systems.

- It is demonstrated that a derivation modelling approach in which requirement models are transformed through a series of modelling aspects involved in coordination and cooperation can be addressed by using what are effectively extensions of current requirements engineering methods.

Limitations of the work presented in this thesis and possibilities for future work are discussed in Chapters 7 and 8.

1.8 Outline of the thesis

Chapter 2 presents the notion of workflow systems to support business processes called business workflow systems, and describes some of the current limitations of these systems that can be related to insufficient requirements engineering methods. Based on this understanding, three needs are formulated for requirements engineering for business workflow systems: the need to use derivation modelling, to consider cooperative work aspects, and to be able to provide methodological support for the requirements engineering process.

In Chapter 3, the aspects of cooperative work are discussed and scenarios are identified as an appropriate means for understanding cooperative work in business workflow situations. A variety of scenario-based approaches are discussed and evaluated. Three main problems are highlighted: the issue of articulation work, visual representations and methodological guidance in scenario-based requirements modelling.

Chapter 4 presents a scenario-based derivation modelling method, which allows modelling and analysing requirements for business workflow systems. The method provides a means of both visualising the behaviour of actors and defining how cooperative behaviour is achieved. The first section describes and explores the problem context and purpose of this approach. Further sections describe its types of models. A standard requirements engineering example, scheduling a meeting, is used to exemplify the approach.

Chapter 5 presents the application of the scenario-based derivation modelling method defined in Chapter 4 to a real-world case study example of complaint management in a bank. As a result, the derivation of articulation protocols from business workflow models define the expectations of the cooperative situation can be fulfilled by the system. Various subtleties were found during the case study, which suggest refinements to the method. The refined method is presented at the end of this chapter.

Chapter 6 describes heuristics for analysis and construction used by requirements engineering practitioners applying the derivation modelling method. These heuristics are a set of rules, which guide the requirements engineering practitioner towards higher rate of success for analysing the available information and for constructing the models. They were derived from experiences and observations in applying the method in both the initial meeting scheduler example and the complaint management case study.

Chapter 7 presents the evaluation of this work. It describes the principal ideas behind the method and indicates major problems found while developing the method and justifying this approach. It reiterates the properties that such a method should display; it then evaluates the work by discussing its weaknesses and strengths with regard to these properties. Finally, this chapter examines the suitability of the visual scenario-based technique and discusses experiences while applying the method to case studies.

Chapter 8 summarises the work of this thesis, revisits the solution as proposed in the first chapter, and suggests possibilities for future work.

2 Requirements Engineering for Business Workflow Systems

This chapter presents the notion of workflow systems to support business processes called business workflow systems, and describes some of the current limitations of these systems that can be related to insufficient requirements engineering methods. Based on this understanding, three needs are formulated for requirements engineering for business workflow systems: the need to use derivation modelling, to consider cooperative work aspects, and to be able to provide methodological support for the requirements engineering process.

2.1 Business Workflow Systems

The workflow concept has evolved from the notion of process in manufacturing and administration. Such processes have existed since the industrialisation and are results of a search to increase efficiency by concentrating on the routine aspects of work activities.

They typically separate work activities into well-defined tasks, roles, etc., which regulate most of the work. Initially, processes were carried out entirely by humans who manipulated physical objects. With the introduction of information technology, processes have been partially or fully automated by software systems, i.e. software programs performing human tasks and enforcing rules which were previously implemented by humans.

Today, workflow management is predominant in a wide range of business and administration tasks (e.g. banking, insurances, or other services). A huge variety of commercial workflow management systems is available to re-engineer, streamline, automate, and track business processes [Sheth and Kochut 1997] [Georgakopoulos et al. 1995]. Market trends (presented by analysts such as Delphi Group or Giga) still show a steady growth of workflow management systems with a lot of potential for applications (e.g. Customer Relationship Management, E-commerce [Muth et al. 1998] [Alonso et al. 1999]). Further, the Internet offers many possibilities and will provide effective and low cost services worldwide to be able to track transactions across enterprise boundaries and to offer services which are adapted to market needs [WfMC 1998].

2.1.1 Business process support

Business processes are descriptions of an organisation's activities implemented as information or material processes. That is, a business process is engineered to fulfil a business contract or satisfy

a specific customer need. Once an organisation captures its business in terms of business processes, it can re-engineer each process to improve it or adapt it to changing requirements. Business process re-engineering might be for increasing customer satisfaction, improving efficiency of business operations, increasing quality of products and services, reducing costs, or meeting new business opportunities by changing existing processes or introducing new ones.

More formally speaking, a business process is "a set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisation structure defining functional roles and relationships." [WfMC 1998]

In the following section, the concept of workflow is highlighted as it is closely related to re-engineering and supporting business processes in an organisation through software systems.

2.1.2 Workflow

A *workflow* may describe business process tasks at a conceptual level necessary for understanding, evaluating, and re-designing business processes. On the other hand, it may describe some tasks at a level that captures requirements for software system functionality or human skills. However, there is a variety of notions around in the literature for workflow. The perspective on the term workflow comes

from the fact that some describe rather business perspectives, others software systems perspectives.

Despite the efforts for standardisation there is still little agreement as to what workflow is. Often, workflow is used casually to refer to a business process, specification of a process, software that implements and automates a process, software that simply supports humans who implement a process. It is used to distinguish workflow specifications from their implementation, or as a collection of tasks organised to accomplish some business processes (e.g. [Georgakopoulos et al. 1995] [Sheth and Kochut 1997] [Hollingsworth 1995]).

Tasks can be performed by one or more humans supported by software systems, or a combination of these. A workflow defines the order of task invocation or constraints under which tasks must be invoked and by whom. Human tasks include interaction with software systems, e.g. providing some input commands or using the system to indicate task progress.

Another characterisation of workflow (which was first given by McCready [McCready 1992]) distinguishes between three kinds of workflows: *ad hoc*, *administrative*, and *production*:

- *Ad hoc workflows* perform office procedures, such as product documentation or sales proposals, where there is no set pattern for moving information among people. Ad hoc workflow tasks typically involve human coordination and cooperation. The coordination of tasks in ad hoc workflow is not automated but is

instead controlled by humans. A typical example may be a meeting scheduler for groups.

- *Administrative workflows* involve repetitive, predictable processes with simple task coordination rules, such as routing an expense report or travel request through an authorisation process. The coordination of tasks can be automated in administrative workflows. In an administrative workflow, users are prompted to perform their task with the support of some software system.

- *Production workflows* involve repetitive and predictable processes, such as loan applications or insurance claims. Production workflow encompasses and involves complex processes, which may access to multiple other software systems. The coordination of tasks in such workflows can be automated. The automation of production workflows is highly complicated due to the process complexity and the exchange of data with other software systems.

Another characterisation of workflow has been presented by Aalst et al. [Aalst et al. 1998]. They divide workflow into structured, unstructured, information centric, and process centric. However, this characterisation does not separate workflow semantics from the software system that supports it.

Yet another characterisation of workflow presents a range from human-oriented to system-oriented [Georgakopoulos et al. 1995]. On the one hand, human-oriented workflow involves humans

collaborating in performing and coordinating tasks. In this case, the requirements for a workflow system is to support the coordination and cooperation of humans. Humans, however, must ensure the consistency of documents and workflow results. On the other hand, system-oriented workflow involves software systems that perform tasks. While human-oriented workflow systems often control and coordinate human tasks, system-oriented workflow systems control and coordinate software tasks (typically with as little as possible human intervention). Consequently, system-oriented workflow systems must include various mechanisms for concurrency control and recovery to ensure consistency and reliability. This in turn is not required by workflow systems that support human-oriented workflow. Human-oriented workflows have process semantics (e.g., capture where to route a document) but have no real knowledge of the semantics (i.e. the information) being processed. System-oriented workflows have more knowledge of semantics (e.g. synchronisation of information).

Consequently, in human-oriented workflow the main issues to be addressed include understanding how people need or prefer to work or how they may interact with the workflow system. In system-oriented workflow, the main issues to be addressed include matching business process requirements to functionality and data, finding appropriate support by the system to perform workflow tasks, and so on.

The research area of CSCW overlaps with workflow issues, where workflow involves predominantly human tasks, as it is the case in

businesses. The development of workflow systems to support cooperative work activities in businesses is faced with challenges similar to those addressed in CSCW, such as dynamic and flexible coordination and cooperation processes.

In the rest of this thesis, the term workflow is used to refer to business processes in which humans participate in organisational activities to achieve a particular goal with the support of software systems.

2.1.3 Workflow management

This work considers *workflow management* as the modelling of business processes. A variety of methodologies have been proposed to carry out process modelling for workflow systems, in particular modelling for business workflow systems. These can be characterised as being *communication-based* or *activity-based*.

- *Communication-based* methodologies are based on "Conversation for Action Model" by Winograd and Flores [Winograd and Flores 1986]. This methodology assumes that the objective of business process re-engineering is to improve customer satisfaction. It reduces every action in a workflow to four phases based on communication between a customer and a performer. In the preparation phase, a customer requests an action to be performed or a performer offers to do some action. In the negotiation phase, both customer and performer agree on

the action to be performed and define the criteria for satisfaction. During the performance phase, the action is performed according to the criteria established. In the acceptance phase, the customer reports satisfaction (or dissatisfaction) with the action performed. Each workflow loop between a customer and a performer can be joined with other workflow loops to complete a business process. The ActionWorkflow Analyst tool [Medina-Mora et al. 1992] from Action Technologies is based on the Winograd/Flores model.

- *Task-based* methodologies focus on modelling the work instead of modelling the commitments among people and do not capture business objectives such as customer satisfaction. Most commercial business workflow systems provide activity-based workflow models (e.g. InConcert or Staffware). These workflow models consist of tasks, whereby each of the task may be comprised from subtasks. Each task has dependencies on the tasks at the same level and has an assigned role, which is the proxy for people of software system to perform the task.

In addition to the above two methodologies, *object-oriented* methodologies, such as the one proposed by Jacobson [Jacobson 1992] may be useful in defining workflow models. For example, Jacobson describes how to identify objects that correspond to actors, to identify the dependencies between those objects, to use object techniques such as inheritance to organise specifications, and to describe *use cases* which are essentially a sequence of tasks

needed to complete some business processes. However, object orientation provides no explicit support for workflow models.

The above workflow management methodologies address the issue of workflow modelling with respect to system orientation, i.e. how the system should be implemented. However, the methodologies do not explicitly support the workflow model of what it means for a workflow to be correct, e.g. what tasks must be completed for the workflow to be considered successful. For example, stakeholders provide the most suitable resource to provide useful and realistic viewpoints on the system for business support. In particular, in terms of business objectives and their impact on the systems supporting the business can be seen as a highly valuable to elicit and validate the requirements of the system to be proposed.

Another problem with these methodologies is that they do no integrate properties such as cooperation or coordination explicitly. The importance and positive impacts of these issues have been addressed in the research area of CSCW.

Finally, workflow methodologies have not addressed different interests and viewpoints. For businesses that rely on their workflow system, modelling requires multiple models for workflow, actors, cooperation between actors, and how the cooperation is achieved.

2.1.4 Workflow management systems for businesses

Several hundred products that provide support for workflow management exist in the market today, focussing on supporting the business environment with emphasis on coordinating human activities, and facilitating document routing, imaging, and reporting [Sheth and Kochut 1997] [Georgakopoulos et al. 1995] [Alonso et al. 1997] [Mohan 1997].

The Workflow Management Coalition defines a workflow management system as "a set of tools providing to support the necessary services of workflow creation, workflow enactment, and administration and monitoring of workflow processes." [Hollingsworth 1995].

Workflow systems can be characterised by the following functional components:

- modelling and representation of workflow processes and their constituent activities,
- selection and instantiation of processes for activation in response to a user request or key events,
- scheduling of activities to agents and resulting tasking of the agents, and
- monitoring and adaptation of executing processes.

The following sections describe critical success factors in designing business workflow systems and some of the limitations of business workflow systems with regard to problems in current software engineering.

2.1.5 Critical success factors in designing business workflow systems

In this section, four major critical success factors in designing business workflow systems are identified in the context of this work. These factors are suitability, adaptability, correctness, and stakeholder involvement and are described below:

- Suitability: a business workflow system must be suitable for purpose of the business. It needs to incorporate structural and behavioural aspects of processes, interactive aspects, and temporal aspects.

- Adaptability: a business workflow system must be adaptable, i.e. it needs to be designed in a way that the system is able to deal with changes. These changes may range from ad-hoc changes (such as changing the order to two tasks for an individual case) to the redesign of a workflow process (may be as part of a business re-engineering project).

- Correctness: a business workflow system needs to provide the necessary correctness and reliability properties in the presence of

concurrency and failures. These aims at both data and workflow consistency in a syntactic and semantic way.

- Stakeholder involvement: adequate stakeholder identification and involvement is a major success factor in designing a business workflow system. Stakeholders are people who are responsible for design and development, people with financial interest, people who are responsible for operations or people who have some interest in its use. However, identifying the right set stakeholders in an organisation-wide workflow implementation of the set of all stakeholders at the right phase of such a project is far from trivial.

The following section describes some of the limitations of current business workflow systems.

2.1.6 Limitations

Due to the number of commercially available business workflow systems, a huge variety of limitations has been documented in the development of workflow systems. Many products have been developed without a clear understanding of user requirements and thus these products are often highly unprepared to meet the demands placed upon them by users embedded in the business. One of the reasons may be that commercial business workflow systems can be traced back to work done in database and distributed systems, system architecture, and transaction systems. Recently,

issues such as scalability, reliability, concurrency control, recovery, high availability, Internet-technology, and interoperability with other system components have been a focus in this research (e.g. [Sheth and Kochut 1997] [Alonso et al. 1997] [WfMC 1998]).

Business workflow systems are systems for supporting coordination and cooperative work. However, observers have argued that business workflow systems make business processes too rigid, not allowing their cooperating users to react freely to the breakdown occurring during their evolution [Bowers et al. 1995]. Some seem to blame the responsibility of this rigidity on their using formal workflow models; others criticise the strict coupling between modelling and executing they introduce (e.g. [Suchman 1987] [Dourish et al. 1996]). An extensive discussion on the pros and contras of this can be found in the papers of Lucy Suchman [Suchman 1994] and Terry Winograd [Winograd 1994].

Clearly, business workflow systems should be oriented towards making businesses as flexible and adaptable as possible and to supporting changes. Furthermore, business workflow systems should allow users to change the flow of work in order to let them handle exceptions and breakdowns without changing it. Business workflow systems should get their flexibility both from the case of dynamically changing them and from not to need continuous frequent changes. Recent research efforts attempt to deal with adaptation and change on a technological basis (e.g. [Aalst et al. 1998] [Koksal et al. 1999] [Sheth and Kochut 1997] [Ellis et al. 1995]). However,

most existing business workflow systems appear to be inadequate with respect to change.

This work takes the assumption that these limitations should not be attributed solely to technological issues, but also to the understanding of the cooperating users and their requirements to work in the business efficiently and effectively.

Another major limitation is the support incorporating aspects of cooperation and coordination. There has been major research efforts in the CSCW area to establish effective software systems support for cooperative work and impressive, but mostly small-scale, research prototypes (e.g. [Bogia and Kaplan 1995] [Schmidt and Simone 1996] [Dourish et al. 1996]). However, little result has been produced as to whether software systems can be successfully designed to support cooperative work for business workflow processes, and so regulate routine coordination activities and thereby enable cooperative actors to perform reliably and efficiently.

Schmidt and Bannon [Schmidt and Bannon 1992] discuss the relevance of articulation work within cooperative work arrangements. Articulation work deals both with the meshing of tasks and performers within a cooperative work process and with the interleaving of different processes within the work time of a performer. Moreover, it deals with the continuous changes of cooperative work arrangements. Therefore, systems supporting articulation work must on the one hand liberate actors as much as possible from the routine articulation work they need for coordination

themselves. On the other hand, systems need to support actors to become aware of the situation where they are performing and to negotiate whenever a breakdown occurs. Finally, they need to be open to continuous change in order of both routines and exceptions.

Existing CSCW systems, such as workflow systems, have been emerging slowly with their progress in gaining widespread acceptance despite growth in network and Internet availability. A number of researchers have presented different arguments for this apparent failure of CSCW systems (e.g. [Grudin 1988]). These issues are not surprising since the main focus of such systems has rarely been the development within a more industrial context. Incorporating an understanding of the nature of the domain has been an issue, which has plagued more traditional forms of systems development. For example, Curtis et al. [Curtis et al. 1988] have identified the lack of application domain expertise as the most significant problem in requirements engineering.

One of the distinctive characteristics in CSCW is the extent in which it is able to focus more on work than being technologically-driven, as many previous approaches to systems development were, and to some extent, still are. A number of researchers have argued for the need to seriously treat the understanding of the nature and the users of cooperative work as part of the development process. Not surprisingly, however, there has been much reticence on how it can be achieved, since this has been done in a highly analytical manner (e.g. [Heath and Luff 1992] [Hughes et al. 1992] [Star 1995]).

2.2 What can Requirements Engineering do?

Before a business workflow process can be modelled, the requirements of the stakeholders need to be understood. This is usually done by interviewing stakeholders with domain knowledge about the process. A variety of methodologies for systems design are used for conducting such interviews to obtain knowledge about the processes.

This traditional way of engineering systems is through conceptual modelling which produces a workflow specification of the system to be developed. The specification concentrates on what the system should do, that is, on its functionality. Such specifications act as a prescription for system construction. This may be done under the assumptions that business workflow systems requirements are stable and given.

However, a number of studies show (e.g. [Lubars et al. 1993] [McGraw and Harbison 1997]) that systems fail due to an insufficient understanding of requirements they seek to address. Further, the amount of effort needed to fix these systems has been found very high (e.g. [Niessink and Vliet 1998] [Ramage and Bennett 1998]). To correct this situation, it is necessary to address the issue of requirements modelling and representation in a utmost focused manner. The expected benefit is that future systems will be more acceptable. The field of requirements engineering has emerged to meet this expectation.

Requirements engineering attempts to go beyond an understanding of what a system does, or should do, to *why* the system is as it is, or should be as proposed. In other words, software systems are seen as fulfilling a particular purpose in the business and requirements engineering supports in conceptualisation of these systems.

Furthermore, requirements engineering considers the potential users of the system as most suitable to provide useful and realistic viewpoints on the system to be developed. Such an exploration leads to the identification of normal and exceptional activities.

An appropriate way of doing requirements engineering must be guided by appropriate models for the problem at hand. Furthermore, as being a complex task, requirements engineering must provide guidance on which activities are appropriate in given situations as well as on how these activities are to be performed.

Taken together, the research results briefly reviewed above suggest that in order to deliver accurate and valid specifications, requirements engineering for business workflow systems must address three main issues:

- the need to have appropriate modelling concepts with cooperation aspects in ensure that purposeful business workflow systems are built,

- the need for the integration of such modelling concepts, and

- the need for methodological support during the requirements engineering process.

These three issues affect both the system to be developed and the process aspects of requirements engineering. However, considering these aspects will result in workflow models that are *suitable* for businesses [Barros et al. 1997]. Suitability implies a close connection between modelling concepts and features, and those required by the particular domain. In terms of suitability for business workflow modelling, it needs the incorporation of often neglected aspects, such as combining structural and behavioural aspects.

2.3 Derivation modelling

Models in requirements engineering serve as a means for communication and validation. Without models, different views on a particular aspect can not be considered. Thus, models help to expose different views, enabling stakeholders to enhance their shared understanding of the phenomenon in question during the requirements engineering process.

The purpose of modelling is to create a clear and structured view of the aspect to be described and helps to restrict to the relevant information and so to account for the separation of concern [Parnas 1972] and problem decomposition [Jackson 1995]. A clear and structured view is not something that can be measured objectively. It

is rather a property that depends on the person for whom and the purpose for which the model is intended. Moreover, this work assumes that visual model representations are highly useful, because of the expressive power they have.

This work has identified the need in Chapter 2.1.3 to use for more than one model for requirements modelling for business workflow systems to understand all aspects of business workflow processes – workflow, actors, cooperation, and articulation between them, and how this is achieved. Moreover, the models need to be derived in a highly flexible manner.

The requirements engineering phase starts from a set of highly informal requirements and may include the capture of the requirements, involving extensive discussions with stakeholders of the future system. Nevertheless, the derivation of various models, from e.g. business workflows, reduces the distance between highly informal and incomplete requirements models and more rigorous methods for designing systems by providing a method which reflects earlier the richness of cooperative work properties as usual. Reducing this distance is one of the main goals of this work, and a primary aim of a derivation modelling method. The results provided by the derivation modelling method is a model, which can then be used as the starting point of a more formal design process.

One approach for creating a derivation modelling method is to indicate heuristics and guidelines to be applied during the requirements engineering process. [4]These heuristics and guidelines

support the analysis, construction, and evolution of the models. Derivation modelling must consider both *in-model* heuristics and *in-between model* heuristics:

- *In-model heuristics* focus on statements for construction and evolution of one particular model.

- *In-between model heuristics* consider statements that support the derivation from one model aspect to another, if there is enough information in the initial model so that the subsequent model can be derived.

Modelling of multiple perspectives and viewpoints has been discussed within the requirements engineering community (e.g. [Nissen and Jarke 1999] [Nuseibeh et al. 1994]. However, these works differ from the above described need for derivation modelling in that they try to derive one valid model from different models covering all the same conceptual aspects. This work, in contrast, attempts to have several models considering the different aspects and merging them at the end.

A large body of work takes a narrower, but much more rigorous approach to model development. This work is not a rigorous or formal refinement or transformation method in the manner of e.g. [Lamsweerde et al. 1995]. It is called a *derivation modelling* method, as it is not formal, but includes a set of notations together with a strategy to be followed and pragmatic heuristics. While formalisation of the approach would be possible, the derivation modelling

approach based on heuristics rather than formal transformation is likely to provide sufficient rigour, appropriate for the human-based subject matter being modelling as suggested by literature on domain knowledge or CSCW.

2.4 Conclusions

This chapter suggests that requirements engineering for business workflow systems is a modelling activity, in which requirements need to be analysed, constructed and evolved.

Four kinds of models are necessary to describe requirements for business workflow systems: workflow, actor, cooperation, and articulation. The essence of the approach described is that these models are concerned with all aspects of business workflow processes.

Three basic needs can be discerned from the above:

- the need to have appropriate modelling concepts with cooperation aspects in ensure that purposeful business workflow systems are built,
- the need for the integration of such modelling concepts, and
- the need for methodological support during the requirements engineering process.

The discussion in Chapter 2.3 argues that the way in which requirements for business workflow systems takes place generates a fourth need as a consequence of the four models (i.e. business workflow, actor, cooperation, and articulation) identified:

- Derivation modelling: a method which produces a set of models that include properties that reflect cooperative properties and can be used as a means for constructing models that can then be used as a better starting point for software development.

These needs are described in more detail and are used to evaluate related work in the following chapter, and are later used to evaluate the work described in the remainder of this thesis.

3 Scenario-based requirements engineering

The need for a derivation modelling method is identified in the previous chapter. In this chapter, the aspects of cooperative work are discussed and scenarios are identified as an appropriate means for understanding cooperative work in business workflow situations. A variety of scenario-based approaches are discussed and evaluated. Three main problems are highlighted: the issue of articulation work, visual representations and methodological guidance in scenario-based requirements modelling.

3.1 Introduction

This thesis is concerned with the activity of modelling in requirements engineering for business workflow systems, and the possibility of reasoning about the models produced. However, this requires a deep understanding of business and human cooperation. Several

approaches have been proposed to address this need for understanding, but typically based on descriptive technqiues.

In this chapter, the properties of cooperative work are explored (Sections 3.2 and 3.3). The use of scenario modelling techniques (Section 3.4) is discussed, and their application to requirements engineering research is reviewed (Section 3.5). This work is evaluated in Section 3.6 and some problems with this work are described.

3.2 Properties of cooperative work

After 15 years of the emergence of the research field of CSCW, researchers still struggle with what CSCW exactly means. Kling [Kling 1992] argues that CSCW may be best characterised as an arena of – and not so much a field of – research. Researchers from multiple research communities actively participate in the CSCW arena, which offers fundamentally new possibilities of computer support for work. Researchers still disagree about the definition of CSCW, though the current emphasis focuses on the first part of the acronym, the computer support. The commercial interest has dramatically increased in products labelled *groupware*, and business workflow system is one category within these.

There is still some disorder about what is meant by cooperative work – the second part of the acronym of CSCW. Since business

workflow systems support cooperative work, it is important to conceive clearly what cooperative work means.

Although a variety of definitions of cooperative work have been brought forward, almost all of them agree on the concept of people working together to achieve a shared goal. Schmidt [Schmidt 1991] characterises cooperative work as a situation "when multiple actors are required to do the work and therefore are mutually dependent in their work and must coordinate and integrate their individual activities to get the work done." Work is always socially situated and socially organised, yet the work process itself is not always intrinsically cooperative in the sense that it requires multiple actors who are thus interdependent in their work [Schmidt 1991] [Hughes et al. 1992]. Studies (e.g. [Luff et al. 1992] [Hughes et al. 1992]) have shown that it is difficult to differentiate an activity as being individual or cooperative.

For a long time, the focus of how people carry out work together has concentrated on positive aspects, such as cooperation, collaboration, and commitment, and disregarded troublesome aspects, such as competition, conflict, and control. Thus, CSCW has been critiqued for being limited in its understanding of cooperative work [Kling 1992]. Workplace studies have shown that an important aspect of work coordination embodies the heterogeneous goals and motives for coordination activities of the different actors [Symon et al. 1996] [Bowers et al. 1995].

The reasons for the existence of cooperative work are multifarious (e.g. [Schmidt 1991] [Bardram 1997]): actors are being able through cooperative work to accomplish tasks that would be infeasible for actors to achieve individually; different viewpoints, goals, motives, heuristics, etc., are integrated by semi-autonomous actors; the manifold ontological structures and representations are *temporary and local closures* [Gerson and Star 1986] and need to be synthesised as part of the cooperative work process.

Actors who are engaged in cooperative work are *mutually dependent* in their work, transforming and controlling an aggregation of interacting objects and processes, often called 'field of work', 'organisational setting', 'work setting', 'system' or 'context' (e.g. [Schmidt 1994] [Malone and Crowston 1994]). Mutual dependence in work means that one actor relies positively on the quality and timeliness of another actor's work and vice versa. Mutual dependence in work can thus be primarily conceived of as a positive, though by no means necessarily concordant, interdependence.

3.3 Articulation work

Cooperative work is distributed in the sense that cooperating semi-autonomous actors have to co-ordinate, schedule, monitor, mesh, integrate, allocate, etc., their individual activities to accomplish an overall task (e.g. to make profit or to satisfy the customer). Sociologists have termed this kind of work *articulation work* [Schmidt 1994].

The concept of articulation work was developed largely by Anselm Strauss [Strauss 1986] and Gerson and Star [Gerson and Star 1986]. In the words of Strauss [Strauss 1986], articulation work is "a kind of supra-type of work in any division of labo[u]r, done by the various actors".

Articulation of cooperative work is essential in multiple aspects: who is doing what, where, when, how, etc.? Therefore, articulation may be expressed in terms of actors, responsibilities, tasks, activities, conceptual structures, and resources. Articulation work is never done in the abstract, but it is always related to the wider context of work environment and organisational setting.

Articulation work is considered as requiring reciprocal awareness through monitoring the activities of cooperating actors or making the activities of one's own activities publicly available to cooperating actors. This may be done by directing attention to other cooperating actors to express a certain state or a potential difficulty, to control activities, etc., by for example pointing, nodding, talking, writing, marking. Articulation work may also include the handing-over of responsibility or assigning a task for a certain process from one actor to another cooperating actor. The articulation of distributed work embodies the use of 'protocols', encompassing a set of explicit conventions and procedures supported by an artefact that stipulates and mediates the articulation of the cooperating actors. Such protocols as characterised by Schmidt [Schmidt 1991] as 'mechanisms of interaction'.

The above approaches have attempted to address the need for understanding cooperative work in a largely descriptive and analytical way. Attempts to use them in software development have had mixed results (e.g. [Bowers et al. 1995] [Grudin and Palen 1995]). One of the reasons being that these has been developed mostly in a research environment rather than in an industrial context.

This work proposes that the understanding of cooperative work can be used in a generative way, as part of the requirement engineering process.

The following sections explore and evaluate scenario-based techniques, its use in requirements engineering and their usefulness for understanding and modelling cooperative work properties.

3.4 Scenarios in requirements engineering

3.4.1 Role of scenarios in requirements engineering

Most requirements engineering methods are built on model-based approaches, ranging from Structured Analysis [Yourdon 1989] to UML [OMG 1999]. These approaches neglect the essential importance of the socially situated context of current and future situations in which the computer system to be developed is used. Rather, they endeavour to establish a complete, consistent and unambiguous requirements specification [Pohl 1994].

Recently, there has been a growing appreciation of *contextualism* in requirements engineering. The term contextualism strives to obtain an understanding of the richness of actors' interactions among themselves or with a computer system in a social context [Potts and Newstetter 1997] [Potts and Hsia 1997]. Contextualism fits in well with approaches such as participatory design or ethnography [Goguen 1994], etc. The synthesis of formal technical and socially situated issues in the practice of requirements engineering is fundamental for building computer systems that work successfully in their social context. The value of such a synthesis has been considered as important just recently in research but is almost non-existent in practice.

In general, requirements are stated in terms of phenomena and relationships that are of interest to the system's stakeholders (manager, user, etc.). Therefore, requirements engineering is concerned with the process of describing requirements for computer systems whose construction is essentially a software development task. Its goal is to provide software that ensures satisfaction of the requirements.

Requirements are located in the environment, which is part of the world, with which the computer system (in the words of Michael Jackson: 'the machine') to be built will interact. The effect of the computer system will be perceived and assessed in the environment. Jackson and Zave [Jackson and Zave 1995] [Jackson 1997] argue that the description of requirements consists of at least two parts – an *optative* (what is desired) and an *indicative* (what is given)

description. An optative description expresses phenomena of the environment that wants to be achieved by installing the computer system. As the introduction of a new computer system usually changes the environment, an indicative description expresses properties of the environment, as they will be when the system is in operation. Unavoidably, the quality of such models depends on the knowledge elicited and modelled from the stakeholders and their successful involvement in the requirements engineering process [Macaulay 1993].

The emergence of object-oriented software engineering [Jacobson 1992] has led to an enormous popularity of scenarios in practice. A recent state-of-practice survey [Weidenhaupt et al. 1998] of scenarios in requirements engineering has revealed that scenarios are used in practice for a variety of reasons. Scenarios

- are used when abstract modelling fails,
- to enforce interdisciplinary learning,
- to require coexistence with a prototype,
- to reduce complexity, in this case scenarios can be considered as a structuring device, and
- to facilitate partial agreement and consistency.

In this thesis, scenarios are considered as an *engine for design* [Mack 1995] during requirements elicitation and validation to

stimulate, facilitate and document shared understanding between stakeholders of both indicative and optative properties – its occurrences, assumptions, action opportunities and risks. The transition from informal to formal is a crucial point in requirements engineering.

Conceptually, cooperative systems can be defined of as three interacting worlds (see e.g. [Kuutti 1995]). The first world is that of a cooperative system consisting of both hardware and software. The second world is that of conceptual analysis and design, which helps to define 'solutions in principle' on a purely abstracted logical level, which has been the focus in traditional requirements engineering. The third world is the real world of work processes, which describes ways in that cooperative systems are used. The use of cooperative systems is always embedded in work processes and becomes meaningful through those work processes. People have also experienced that if a new computer system is introduced in an environment, the situations change. It has become evident over the past few years that those situations must be explicitly studied as well.

Software engineering research has recognised the need to deal with third world issues. Christiane Floyd's [Floyd 1987] seminal paper 'Outline of a paradigm change in software engineering' contrasts two different perspectives in software engineering: a product-oriented view and a process-oriented view. The product-oriented view regards software systems as a product standing on its own, consisting of a set of programs and related defining texts. In doing so, the product-oriented view abstracts from the characteristics of the given base

machine and considers the usage context of the product to be fixed and well understood, thus allowing software requirements to be determined in advance. The process-oriented view considers software in connection with human learning, work, and communication, taking place in an evolving world with changing needs. Processes of work learning and communication occur in both software development and use.

In contrast to purely model-based approaches, scenarios offer a sufficiently deep *middle-level abstraction* [Carroll 1995] between models and reality, promoting a shared understanding of contextual properties of an existing system and its future system requirements.

Recently, some researchers have started to recognise the need to make the goal hierarchies driving a scenario-based requirements engineering process explicit [Antón 1996] [Dardenne et al. 1993] [Lamsweerde et al. 1995] [Yu and Mylopoulos 1994]. The combination of these two extensions has also been conceived as highly relevant for guiding change management [Haumer et al. 1998].

3.4.2 Definitions of a scenario

Despite the popularity of scenario-based approaches, there is no generally accepted definition of what a scenario is, what it should entail, or how it should be used. The definition of Carroll [Carroll

1995] seems to be a good outset of what a scenario for requirements engineering of cooperative systems should cover:

"The defining property of a scenario is that it projects a concrete description of activity that the user engages in when performing a specific task, a description sufficiently detailed so that design implications can be inferred and reasoned about. Using scenarios in system development helps keep the future use of the envisioned system in view as the system is designed and implemented; it makes use concrete..." (p. 3-4).

Unfortunately, this broad definition does not provide answers on the novelty of scenario descriptions in contrast to traditional requirements specifications. It neither gives an answer how concrete the description of activities should be nor what an activity represents in this context. Furthermore, the definition fails to explain what implications are meant and what the role of the computer system plays in the scenario description. Finally, it is unclear how a scenario in form of textual representation can help the envisioning of a future use situation.

A variety of scenario-based approaches have their origin in the human-computer interaction area and have thus purely concentrated on the interaction between a computer system and a user, but rarely on actual work situations in the environment context.

Most recently, members of the CREWS project [Jarke et al. 1999] have aimed at developing a definition based on the current understanding in both research and practice:

"A scenario is a description of the world, in a context and for a purpose, focusing on task interaction. It is intended as a means of communication among stakeholders, and to constrain requirements engineering from one or more viewpoints (usually not complete, not consistent and not formal)."

However, none of the above definitions is entirely satisfactory in the context of this thesis. Requirements engineering methods for supporting cooperative systems development must be concerned with modelling the interaction of cooperating actors who have to articulate their individual activities to accomplish an overall task. Such methods inherently enable us to understand socially situated and socially organised cooperative work processes.

3.4.3 Scope of scenarios

Scenarios may be categorised according to the scope they address:

- *Internal system scenarios* describe the interaction between internal system components without consideration of external context of the system.

- *Interactional scenarios* describe the direct interaction between the system and the actors of the environment and express constraints, which the environment places on the system. Those kinds of scenarios are the most frequent approaches found in research and practice. The main reason for this may be that

since the late 1980s researchers in the area of human-computer interaction (HCI) have used scenarios as tool for eliciting and representing system requirements to improve communication between system developers and stakeholders. In the past few years, interaction scenarios have gained enormous popularity in particular through Ivar Jacobson's approach [Jacobson 1995], which has also fed into the efforts to establish a Unified Modeling Language (UML) for systems engineering based on the object-oriented approach. Other examples of interaction scenario-approaches can be found in [Nielsen 1995] [Nardi 1995] [Cockburn 1997] [Potts et al. 1994].

- *Contextual scenarios* describe the interaction between the environment and between the system and its environment. They consider the organisational work context [Kyng 1995] including issues such as goals, resources, business processes, etc., based in the environment. This approach is reflected in the *participatory design* area, acknowledging that an explicit and active involvement of stakeholders in the design process constitutes good computer support in their context of work. In addition, research in CSCW (e.g. [Suchman 1987] [Rogers and Ellis 1994] [Bowers et al. 1995] [Jordan 1996] [Star 1995]) has convincingly revealed that supporting work context as it is actually done in real life calls for a more intrinsic understanding and description of work processes.

Since this thesis is concerned with cooperative work processes, a method must at least be able to represent contextual issues independently of a computer system. This is in the same vein as Jackson's world and machine approach – adequate elicitation and analysis of requirements starts with modelling the environment and successively changing the model by identifying new indicative and optative properties.

3.4.4 Representation of scenarios

Scenarios are very often represented using informal or semi-formal text. One advantage of text is to express a problem in a comprehensible way [Karat 1995].

However, conclusions from industrial case studies of scenarios in requirements engineering indicate that text representation is insufficient and graphical representations are suggested for representing scenarios [Weidenhaupt et al. 1998].

3.4.5 Bridging the gap between CSCW and requirements engineering

With the advent of groupware products, CSCW has evolved as a research arena separated from traditional software systems engineering but is now bridging the existing gap. Indicators for this development are found in an increasing number of papers (e.g. [Potts and Newstetter 1997] [Goguen 1994]) that both must evolve to

accommodate the strengths of each other to produce effective methods of deriving requirements for computer systems that support cooperative work. Nevertheless, there is still a gap between these two disciplines.

The suggestion of this work is to bridge the gap by developing a method as a means to integrate the research results found in CSCW into requirements engineering. A scenario-based approach seems to offer the potential as a basis for achieving this.

The issues in the previous sections provide bounds to the following review of scenario-based approaches, and are addressed explicitly in the subsequent evaluation section.

3.5 Scenario-based approaches for requirements engineering

The principal approaches of interest in this thesis are those that

- allow organisational work context to be expressed,
- provide support for the construction and evolution of scenarios, and
- serve some degree to describe cooperative behaviour.

This section reviews the most prominent approaches whose underlying representation is textually and graphically oriented.

Structured or semi-structured text using natural language is the underlying representation of most scenario-based approaches in requirements engineering. Studies such as [Rolland et al. 1998a] or [Weidenhaupt et al. 1998] have shown that in both research and industry more than a dozen scenario-based approaches suggest the use of natural language. Its popularity comes from the fact that natural language provides a way to express problems in a relatively easy to understand representation (e.g. [Kyng 1995]).

Jacobson [Jacobson 1995] has developed a use case approach that is essentially a narrative informal description of use, responsibilities and services within the object-oriented area and aims to support the capture of system requirements. Use cases are expressed in entity types, like customer or supplier. In his context, a scenario is a use case instance with concrete actor names, event parameters, states, and conditions.

Use cases are centred around behavioural requirements. Jacobson's approach allows only the interaction between the system and its environment to be covered. Organisational information (such as goals or non-functional requirements (e.g. performance)) or system internal issues that may not be observed by the user are excluded from the description.

The use case approach of Jacobson provides only modest methodological guidelines for constructing scenarios in the form of sequences of tasks to be carried out: find actors, find use cases, prioritise use cases, describe use cases, select metrics, review.

Methodological rules for situations, alternative ways of working, etc., are not taken into account within the use case approach.

Some researchers have made proposals to extend the use case approach. Cockburn [Cockburn 1997] suggests the concept of 'goal' as an important element of use cases. Used as a structuring mechanism for use cases, every interaction between an actor and the system is connected to a goal assigned to either an actor or the system (which is basically an actor, too). Interactions between an actor and the system end when a goal is delivered or abandoned. Therefore, a use case is discovered each time a goal is discovered.

Regnell's approach [Regnell et al. 1995] [Regnell 1999] is an extension of the use case driven approach proposed by Jacobson [Jacobson 1995]. However, the use case description uses a more formal notation. Use case specifications are used to refine use case descriptions. Descriptions use events, condition and problem domain objects as the underlying concepts. Specifications use time, atomic operations, and abstract interface objects as the underlying concepts. Structured text and graphical representation are used to describe use case descriptions and specifications. Regnell's approach allows to describe both interactional scenarios (as in Jacobson's approach) and internal system scenarios. Internal system scenarios are captured at the level of atomic operations.

Rolland et al. [Rolland et al. 1998b] have proposed an approach which allows the coupling of intentional and operational descriptions in the form of goals and scenarios. Their aim is both the refinement

and the discovery of goals. The process is centred around the notion of a 'requirement chunk', which is a pair of a goal and a scenario. Requirement chunks are constructed through composition, alternative, or refinement relationships. The composition relationship links requirement chunks that are required defined a complete requirements model. The alternative relationship characterises some alternatives to reach the same goal. Refinement is used to describe requirement chunks at different levels of abstraction and is therefore used as a mechanism to hide details in order to focus on essential aspects. In general, such relationships, which have formerly been suggested in requirements engineering (see e.g. [Dardenne et al. 1993] and [Yu and Mylopoulos 1994]), aim to support the exploration of alternatives and completeness and the refinement of requirements. Scenarios are represented using semi-structured text based on formal semantics clauses [Rolland and Achour 1998] .

Potts et al. [Potts et al. 1994] propose a requirements analysis model called Inquiry Cycle which aims to support the documentation, discussion and evolution of requirements. Scenarios are represented in textual form following some tabular notations and are expressed at the instance level referring to specific agent names or events with concrete argument values. In situations where entities such as agents of the same type interact, or when several entities of one type interact with an entity of another type, it may be beneficial to have entity instances to avoid confusion. Hence, the use of concrete scenarios is intended to reduce ambiguities. In the example of Potts et al., the initiator, Esther, has scheduled a particular meeting that requires that attendance of Annie (active), Kenji (important) and Colin

(ordinary). The meeting must be held next week, but Kenji and Colin can attend only on days when Annie is out of town. The brief example show that dealing with instances helps to understand why no meeting is feasible.

The requirements engineering process within the Inquiry Cycle is supported by a hypertext tool in which scenarios and requirements are annotated with requirements discussions, rationales, and change requests.

The Inquiry Cycle allows the requirements engineer to take into account both internal system scenarios and system interactions. However, two main criticisms can be levelled at the Inquiry Cycle:

- contextual scenarios are utterly excluded, and
- no support of how to construct scenarios in the Inquiry Cycle is provided.

The problem for contextual scenarios rest with the use of goals, work situations, etc. Kyng's [Kyng 1995] scenario activities are organised in a five-step approach, each based on a specific type of scenarios that captures organisational context. The approach includes functional, non-functional and intentional aspects. Scenarios are represented through informal text. Bardram [Bardram 1998] suggests a similar approach. He distinguishes four kinds of scenario descriptions: organisational, person-oriented, object-oriented, and setting-oriented. Both approaches are solely text-based.

3.6 Evaluation

Each of the above approaches has its strengths and weaknesses. The following evaluation, which is essentially informal, is carried out with regard to each of the issues given in Sections 3.2, 3.3, and 3.4, concerning the nature of cooperative work and the use of scenarios in requirements engineering. The final section concludes with a summary of what can be learnt from these approaches.

3.6.1 Articulation work

The distinction between cooperative work and articulation work is fundamental, because cooperating semi-autonomous actors have to articulate their individual activities to accomplish an overall task. The distinction should be made in scenario-based requirements engineering approach.

Jacobson's approach expresses relationships between actors but lacks expressiveness of how actors co-ordinate themselves in their tasks. Regnell's use case descriptions describe the articulation work done by one actor implicitly when modifying the work objects and processes.

An action triggers an interaction with an another actor in the approach of Cockburn. The interaction may allow achieving the actor's goal by calling the responsibility of the regarding actor.

However, the approach does not make explicit the variety of expressions necessary to articulate distributed activities.

Kyng and Bardram both emphasis the issue of cooperatively working actors who need to co-ordinate their work and thus provide some rudimentary descriptions about articulation work (what is done? where and when is it done? who is doing it? why? and how is it done?). However, they are not made explicit and therefore difficult to distinguish from the actual cooperative work.

3.6.2 Articulation protocols

Articulation protocols arrange the articulation of activities among cooperating actors through artefacts. The state of the protocol is distinct from the state of the underlying work processes.

Bardram's work activity scenarios describe mechanisms of interaction in the form of who is doing what, when and why, distinguishing how it is done today (maybe without a computer system) and how it may be done with some cooperative system.

3.6.3 Methodological support

Requirements elicitation methodological guidelines for the construction and evolution of scenario descriptions are crucial, because some instrument to control the level of granularity of

scenarios is essential. Cockburn [Cockburn 1997] does not suggest methodological guidelines of how to associate goals with scenarios or how to track goals. Regnell's [Regnell et al. 1995] approach lacks methodological guidelines.

In contrast to step-by-step methods Roland et al. [Rolland et al. 1998b] suggest a non-linear method: it allows the process to start at each goal discovered so far. A variety of guiding rules can be applied at different levels of abstraction. The difficulty with this approach is the degree of detail at each goal.

Kyng [Kyng 1995] provides a high-level process, but gives no justification of the stepwise activities and the specific types of scenarios.

3.6.4 Contextual description

Jacobson's approach [Jacobson 1995] covers only the interaction between the system and its environment.

The approaches of Regnell [Regnell et al. 1995] and Potts et al. [Potts et al. 1994] allow both the description of interactions between the system and its environment, and internal system interactions.

Both Kyng and Bardram include in their scenarios organisational work context.

3.6.5 Visual representation

Requirements modelling is usually done together with stakeholders. Visual representations of scenarios are easier to understand than purely textual representations. Jacobson uses a very simple graphical representation beside a textual one to describe relationships between actors. Regnell uses some sort of flow diagram expressing user and system actions. Both approaches do not allow goals etc. to be expressed.

The other approaches mentioned in section 3.5 use textual representation only.

3.6.6 Summary

As a result of the above evaluation, it can be concluded that three of the key features to be provided by such a method for business workflow systems supporting cooperative business processes are:

- the distinction between articulation work and cooperative work (Chapters 3.3 and 3.6.1),
- an ability to express articulation protocols for the cooperative work process (Chapter 3.3 and 3.6.2), and
- a visual-based representation to provide smooth understanding for communication and discussion with stakeholders (Chapter 3.4.4 and 3.6.5).

Visual representations have been investigated in the area of describing distributed processes for telecommunication systems with Use Case Maps [Buhr and Casselman 1995] [Buhr 1998]. These techniques seem to have potential to address issues that have not adequately addressed by current work.

3.7 Remainder of the thesis

In this chapter, scenario-based methods are reviewed and evaluated in Sections 3.5 and 3.6.

Chapter 4 presents a derivation modelling method, which allows modelling and analysing requirements for business workflow systems. The method provides a means of both visualising the behaviour of actors and defining how cooperative behaviour is achieved.

Chapter 5 presents the application of the method defined in Chapter 4 to a real-world case study example of complaint management in a bank. The business workflow model, the actor model, the cooperation model, and the articulation model of the example are systematically explored, using the method.

Chapter 6 describes heuristics and guidelines to support the requirements engineering practitioner to use the derivation modelling method.

Chapter 7 presents the evaluation of this work and Chapter 8 summaries this work.

4 Scenario-based Derivation Modelling in Requirements Engineering for Business Workflow Systems

This chapter presents a scenario-based derivation modelling method, which allows modelling and analysing requirements for business workflow systems. The method provides a means of both visualising the behaviour of actors and defining how cooperative behaviour is achieved. The first section describes and explores the problem context and purpose of this approach. Further sections describe its types of models. A standard requirements engineering example, scheduling a meeting, is used to exemplify the approach.

4.1 Introduction

This chapter is mainly devoted to the provision of concepts within a method for modelling, analysing and communicating requirements for the design of business workflow systems. A visual scenario-based technique is used to give a bird's eye view of the system as a whole

and to provide a starting point for developing specifications to satisfy requirements.

The main novel aspect of this work is that it encourages a *scenario-based derivation modelling* approach in which requirement models are developed through a series of transformations, in which humans can manipulate models in order to derive other models. The further novel aspect is that an articulation model is explicitly derived in form of a requirements model, so that stakeholders can understand how cooperative work can be achieved through the system to be developed.

The approach starts with a business workflow modelling phase. The goal of this phase, apart from modelling the actors and their relationships, is to produce models that capture the high-level workflow structure and behaviour. Further modelling provides an actor model, a cooperation model, and an articulation model. The goal is to have a clear understanding of the behaviours, the actors that cooperate, and their dependencies, as well as the further activities to coordinate their interdependent activities.

4.1.1 Problem context

The past ten years have seen the application of sociology (e.g. ethnography) become increasingly more prevalent in both of the research areas, requirements engineering and CSCW. A variety of studies have been performed in diverse domains, including

underground control rooms [Heath and Luff 1992], air traffic control [Hughes et al. 1992], and hospital work [Symon et al. 1996] [Bardram 1997]. These studies have uncovered delicate facets of the social character of cooperative work that are central to the successful functioning, although these facets seem to be so trivial that traditional methods of requirements engineering may fail to notice them. However, sociological techniques, such as ethnography, are limited in an industrial environment, because they are time-consuming, produce textual descriptions, and lack a methodical approach.

This work represents an approach to integrate some of the important results on articulation work in the CSCW research area within the field of requirements engineering. In particular, the method draws on results from the sociologists, Anselm Strauss [Strauss 1986], Elihu Gerson and Susan Leigh Star [Gerson and Star 1986], and Kjeld Schmidt [Schmidt 1994].

To understand the nature and character of the development of business workflow system, the first thing to do is to look at the business workflow system context. In the words of Jackson [Jackson 1997], the context is made up of those parts of the world that affect the system and are affected by it; the parts of the world that you would eventually look at to judge whether the system is fulfilling its function and serving its purpose successfully.

Business workflow systems can be found in many areas of administration and business with a range of different brands and

purposes, though in which the pivotal function and purpose of the system is to support the work of cooperating actors within an organisation or between an organisation with the environment, such as suppliers, customers, etc. Recently, business workflow systems have been discovered as a platform for electronic commerce applications [Muth et al. 1998].

4.1.2 Purpose

This work assumes that requirements engineering for different systems calls for different methods. The characteristics and scope of a method must be adapted to the characteristics and scope of its context, of the functions it serves, and the problems it solves. The development of a method that is suitable for the development of business workflow systems requires an understanding and an identification of the nature of cooperative work. It further needs understanding of how business workflow systems are used in such situations. The main characteristics have been identified in Chapters 2 and 3.

This method aims primarily to improve modelling, analysing, and communicating user requirements among stakeholders. The practicality of the approach was confirmed by the application of the method to a real-world case study (Chapter 5).

4.1.3 Overview of the method

Four types of models are used within this approach during the requirements engineering process (Figure 1):

- The *Business workflow model* identifies actors and their behaviour. It gives a high-level view of the actors and workflows, and provides a starting point for deriving the details of the other models. It is generated by tracing workflow scenarios that describe tasks, actors, and their behaviour along the way.

- The *Actor model* describes the behavioural structure of the actors discovered in the business workflow model. The actor model is derived from the high-level workflow model and is described in terms of their goals and tasks.

- The *Cooperation model* describes actor relationships in terms of cooperation dependencies.

- The *Articulation model* describes what *articulation protocols* need to exist for actors to cooperate with each other using a business workflow system. The articulation model is derived from the actor model and the cooperation model. It defines what articulation protocols need to exist to fulfil the dependencies identified in the cooperation model.

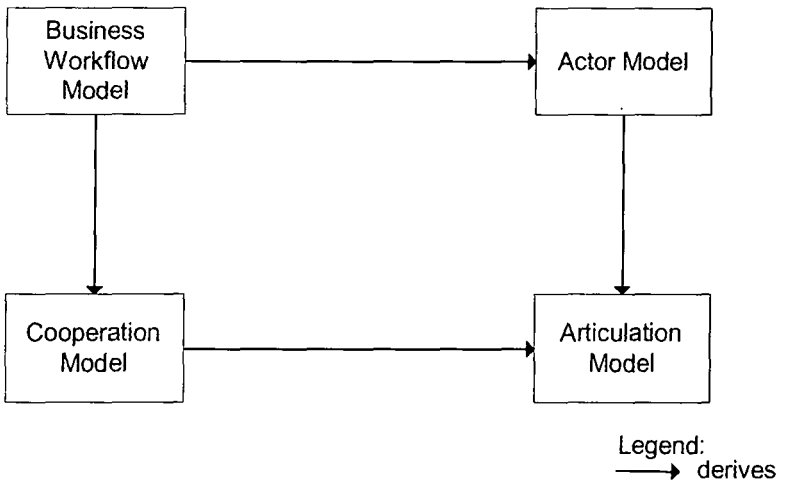


Figure 1: Produced models of this approach

The above four models satisfy the need for modelling concepts with cooperation properties. The use of derivation rules supports the integration to each other.

4.2 Business workflow model

The *Business workflow model* identifies actors and their behaviour. It gives a view of the actors and workflows, and provides a starting point for deriving the details of the other models. It is generated by tracing workflow scenarios that describe tasks, actors, and their behaviour along the way.

4.2.1 Modelling workflow behaviour

The aim of business workflow model is the modelling of requirements that leads to a high-level view of actors and their behaviour in a first step. One of the main goals is the need to describe system boundaries. These boundaries define, at a high-level, where the final delivered system will fit into the current operational environment. Identifying a system's boundaries affects all subsequent modelling efforts.

The result of the requirements modelling phase is

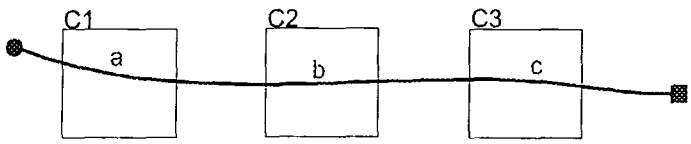
- the definition of operational aspects of the model, such as tasks of actors, and system changes caused by the performance of some tasks, and

- the macroscopic behaviour at the level of cooperating actors achieving some specific purpose supported by a system.

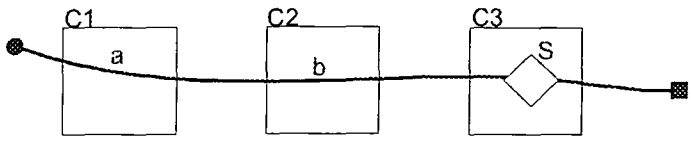
4.2.2 Use Case Maps

Use Case Maps (UCMs) [Buhr and Casselman 1995] [Buhr 1998] are precise structural entities that enable the description, in a high-level way, how the organisational structure and the emergent behaviour are intertwined. UCMs provide a notation that helps humans to visualise, think about and explain the big picture in terms of causal sequences in form of paths. Causal sequences are called scenarios.

In general, UCMs may have many paths. However, for simplicity reasons, the example in Figure 2 shows only one path. The causality expressed by the paths is understood by humans due to the visual nature of UCMs.



a) Example of a UCM



b) UCM with a stub

Figure 2: Examples of UCMs

A filled circle indicates a *start point* of a scenario path, the point where stimuli occur causing an activity to start progressing along the path. A bar indicates an *end point*, the point where the effect of stimuli is felt. Paths trace causal sequences between start and end points. The causal sequences connect responsibilities, indicated by name points along paths (Figure 2a). Paths are superimposed on boxes representing operational components (e.g. *C1*, *C2*, and *C3*), to indicate where components participate in the causal sequences. A component may be a human or system actor. Individual paths may

cross many components and components may have many paths crossing them.

The basic assumption is that stimulus-response behaviour can be represented in a simple way with paths. This is a very common characteristic of business workflow situations which is also of concern in this thesis. The result is a path-centric view rather than a conventional actor-centric (i.e. component-centric) view.

UCMs may be decomposed using a generalisation of responsibilities called *stubs* (e.g. S in Figure 2 b). Stubs may be positioned along paths like responsibilities but are more general than responsibilities in two ways:

- they identify the existence of sub-UCMs, and
- they may span multiple paths.

Stubs enable to draw UCMs that give a high-level overview of the general trend of paths, while leaving details that might obscure the big picture to sub-UCMs shown in separate diagrams. A plug-in may involve additional components not shown in the main UCM.

More features of UCM are described in the Appendix B.

UCMs are used to model the business workflow activities for the following reasons. They are

- able to simply and successfully depict the model of complex systems, and

- provide a powerful visual notation for review and detailed critique of the model.

4.2.3 Deriving the business workflow model with scenarios

This work uses UCMs in a scenario-driven approach for the description of business workflow system requirements. It is intended to bridge the gap between "early" informal requirements and a first high-level design and thus to improve the maturity of the requirements engineering process.

The business workflow model can be derived by tracing scenarios describing functional behaviour as paths. This leads to identifying actors and responsibilities (responsibilities can be considered as tasks performed by actors), and stubs along the way. Generally, one starts with some scenarios and some knowledge of the actors required realising them. However, there is no requirement that all actors or all scenarios are known beforehand. One may start from very general ideas about both scenarios and actors. For example, UCMs may be used to elicit actors to realise paths that represent scenarios, or they elicit new paths that traverse known actors.

The intention of this scenario-driven strategy is to produce a first business workflow model. It is aimed to define a set of scenarios as

complete and consistent as possible. The goal is to produce aggregated, closely related, scenarios ("scenario clusters") instead of individual and sequential scenarios ("traces"). This provides alternative outputs to the same input and one valid and several exceptional scenarios. Further, the composition of multiple scenarios into one is simplified by the visual nature of UCMs. However, this work does not try to manage all aggregated scenarios together to synthesise a global model.

4.2.4 Derivation rules

The steps involved in the business workflow modelling phase can be summarised as follows:

- identify scenarios and major components involved,
- draw paths that connect the identified components,
- flush out the scenarios by identifying more components and their roles,
- identify precondition and postconditions for each scenario,
- identify responsibilities and constraints for each component in a scenario, and
- identify responsibilities that can be achieved by different sub-scenarios and replace them with stubs.

The following section provides a case study example of how UCMs can used of workflow modelling for business workflow systems.

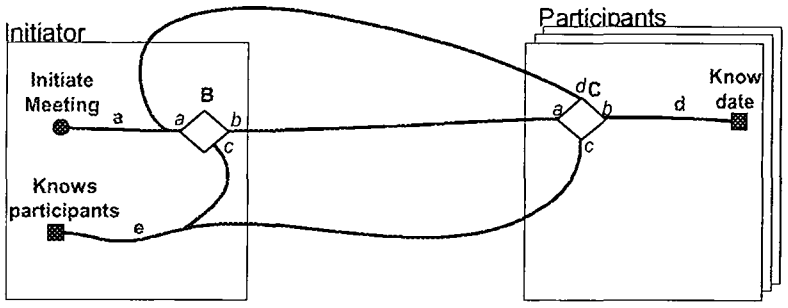
4.2.5 Example: Scheduling a meeting

This section gives an example of a business workflow model with the use of UCMs. The example is based on the meeting scheduler problem of Lamsweerde *et al.* [Lamsweerde et al. 1992] and Potts *et al.* [Potts et al. 1994]. The meeting scheduler problem can be seen as a typical business workflow problem and has been treated as a research benchmark tool in requirements engineering research [Feather et al. 1997] in recent years.

A meeting scheduler system supports people to schedule rooms and equipment for meetings. A meeting is requested by an *initiator* and it may have two or more *participants*. The initiator proposes some time constraints for the meeting, and the potential attendees respond with their available and preferred times, location and/or equipment requirements.

This section does not intend to provide a full specification of the meeting scheduler system requirements but focuses to demonstrate some of the important features to produce a business workflow model.

Figure 3 shows a UCM for a basic scenario of scheduling a meeting. The precondition of scheduling a meeting is that an initiator wants to schedule a meeting.



Reponsibilities and stubs:
a. Define preference and exclusion sets
B. Process sets, check participants list
C. Process sets
d. Inform on date
e. Direct attention on status

Precondition:
Meeting initiator invites for a meeting

Postconditions:
Initiator informed on who attends
Participants know meeting date

Figure 3: High-level workflow model of scheduling a meeting

The scenario shows two actors, the meeting initiator at the left and the stack that describes one or more instances of participants at the right. However, the initiator and the participants are two distinct roles, but the initiator may also be a participant of the meeting. The two roles are separated to make the high-level workflow easier to understand.

The scenario path begins with responsibility *a*, where the initiator defines preference and exclusion sets for the meeting to be scheduled. After the definition of the preference and exclusion sets, the path leads to stub *B*, which processes the defined sets and also checks the list of participants for the meeting. Stub *B* has two outgoing ports, *b* and *c*. Port *b* is followed when the participant exists

and can be invited for the meeting. Port *c* is followed in case the participant does not exist and the meeting initiator is informed about this (responsibility *e*).

In the case each of the invited participants exist (port *b* is followed), the path leads to stub *C*, which processes the sets to each of the invited participants. Stub *C* has three outgoing ports, *b*, *c*, and *d*. Port *b* followed only if the all the participants accept the proposed date. Port *c* is followed to inform the meeting initiator about scheduling status. The scheduling status informs the meeting initiator if the invited participants can attend the meeting, if they accepted or refused the invitation. An example of situations when a participant refuses the invitation is when the participant has already scheduled another meeting. Port *d* is the means by which a participant and the meeting initiator negotiate on the preference or exclusion sets. If the negotiation between the meeting initiator and the participants find an acceptable date, the path leads to responsibility *d*, which means all participants are informed on the meeting date. The postcondition of this scenario is that the participants know the meeting date.

Figure 4 illustrates two plug-ins for the decomposition of stubs. In these plug-ins, the points from which the main path continues are labelled.

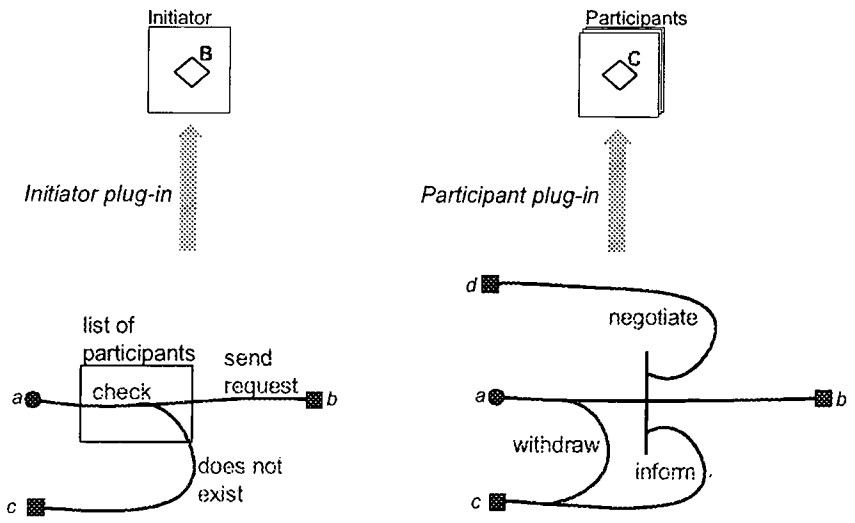


Figure 4: Plug-ins for scheduling a meeting

Stub *B* can be decomposed to the following: The path begins by checking the participants list. If the participant is in the list, the request is refused. This is illustrated by the fork in the path that follows the *check* responsibility. The simple fork in the path immediately after the *check* responsibility is called an *or-fork*, and indicates alternative scenario paths. Otherwise, the meeting request is sent to the participants.

Plug-in for stub *C* decomposes behaviour into the following: The plug-in starts with an or-fork. If the participant cannot accept the invitation for a particular reason, the path labelled *withdraw* is followed and the initiator is informed that the participant has withdrawn from the meeting. Otherwise, the path is forked into three concurrent paths. The fork, with the bar across it, is called an *and-fork*, and indicates that the scenario proceeds concurrently along

three paths. One fork allows the participants to know the proposed preference and exclusion sets of the meeting. The second negotiates with the meeting initiator in case of conflicts. The third informs the initiator on the status of the participants.

4.3 Actor model

The *Actor model* describes the actors and their behavioural structure discovered in the business workflow model. The actor model is derived from the business workflow model and is described in terms of their goals and tasks.

In the following sections, the derivation mapping and rules are described.

4.3.1 Mapping

The mapping from UCMs to actor models elements is as follows (see Table 1):

USE CASE MAPS	ACTOR MODEL ELEMENTS
Path segments that traverse an actor	Goal column
Responsibilities	Actor task
Stubs	Set of tasks
Preconditions	Preconditions
Postconditions	Postconditions
Path segment connecting two actors	Interaction

Table 1: Derivation mapping UCM to Actor model elements

Path segments that traverse an actor represent goals, static stubs represent sets of actor tasks, path preconditions and postconditions help to form preconditions and postconditions. Responsibilities along the path constitute the actor's tasks. In addition, the model captures, if needed, the causality relationship in business workflows. This is done by converting path segments connecting two actors in a UCM to tasks in the actor model. Each of these tasks is basically responsible for causing tasks for other actors to be started.

The actor model is represented in tabular form with five columns:

- The goal column lists goals an actor wants to achieve.
- The precondition column lists conditions that must hold in order for goals or tasks to be performed.
- The postcondition column lists the effects of performing a successful goal or task.
- The task column lists all the actor tasks, including subgoals that are required to fulfil each goal. A goal may be decomposed into subgoals, which provide detailed or alternate ways of achieving that goal. These subgoals are shown in the task column as well as in the goal column.
- The comment column contains a textual explanation.

If a path segment has responsibilities or more than one stub, then the path segment should be mapped to a goal in the actor model.

UCMs allow different scenarios to share a common path segment. Sometimes the only thing these scenarios have in common is the responsibilities along the common path segment. A requirements engineer must decide if a common path segment should be mapped to one goal for all scenarios or to a goal for each scenario.

4.3.2 Derivation rules

The process of building the actor model from business workflow models can be summarised as follows:

- Analyse each path segment that traverses an actor and associate a goal with it and tasks with its responsibilities.

- For each path segment, identify preconditions and postconditions and map them to preconditions and postconditions in the actor model.

- Analyse path segments that connect actors and identify actor tasks that are responsible for causing tasks for other actors.

Tables are used to describe an actor model, where for each actor exactly one table is built.

4.3.3 Meeting scheduler example continued

This section continues the examples of scheduling a meeting from Section 4.2.5. In order to derive the actor model, the different UCM path segments that cross them need to be examined.

In the scenario in Figure 3, it is shown that there is one path segment for the initiator and one for the participants that cross them. Each of these segments is mapped to a goal and inserted into the actor model, as shown in row 1 in Table 2 and in row 1 Table 3.

The preconditions and postconditions of each path segment are inserted in the corresponding row. Stubs along paths are inserted as tasks in the tasks column. Each of these stubs is mapped into tasks in the actor model. Responsibilities for each plug-in are captured in the task row.

The actor model for the meeting initiator is presented in Table 2.

GOAL	PRE	POST	TASK	COMMENT
Initiate meeting	Meeting initiation decided	Meeting request transferred to participant or participant not allowed	Define preference and exclusion sets Check list of participants Send request Refuse participants	Initiator in main UCM and plug-in
Inform on status	Status changed	Initiator informed on status	Direct attention on status	Initiator in main UCM

Table 2: Actor model for meeting initiator

The actor model for the meeting participant is presented in Table 3.

GOAL	PRE	POST	TASK	COMMENT
Process initiation	Meeting is initiated	Participant is withdrawn or negotiates or accepts the meeting request	Withdraw from meeting Negotiate about date Accept meeting request	Initiator in main UCM and plug-in
Inform on date	Date decided	Participant informed and date is known	Inform on meeting date	Initiator in main UCM

Table 3: Actor model for meeting participant

4.4 Cooperation model

The *Cooperation model* describes actor relationships in terms of cooperation dependencies. It describes cooperation dependencies between actors (either human or system). A dependency relates an actor that provides a service to an actor that requires that service. An example of dependencies are goals to be achieved and tasks to be performed.

This work has identified five types of actor dependencies: goal, task, resource, state, and interaction dependencies. The goal, task and resource dependencies are similar to the dependencies described by Eric Yu [Yu and Mylopoulos 1994] for capturing numerous kinds of constraints and relationships that are frequently encountered in business processes.

The following gives a description of these dependencies:

- *Goal dependency* indicates that an actor is dependent on another actor to achieve a certain goal. However, the dependent actor does not specify how the other actor should fulfil the goal.

- *Task dependency* indicates that an actor requires a specific task to be performed.

- *Resource dependency* indicates that an actor is dependent on a supplying actor to provide it with a specific resource.

- *State dependency* indicates that an actor is dependent on another actor to direct attention to a particular state.

- *Interaction dependency* indicates that an interaction is required to fulfil the dependency. The identification of these dependency types helps in choosing the interaction.

Figure 5 illustrates the different symbols used in the cooperation model. A dependency is shown in the cooperation model diagram as an arrow going from a dependee (i.e. a supplier) to a dependent actor. The figure illustrates the five types of graphical symbols used to differentiate dependencies.

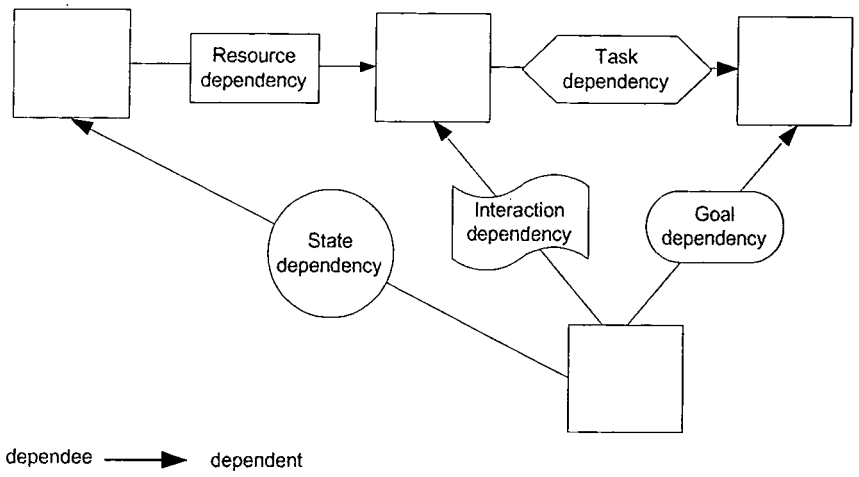


Figure 5: Symbols used in cooperation model

4.4.1 Deriving the cooperation model

The cooperation model is derived from the path segments in the business workflow model that connect two actors, i.e. where two actors cooperate. Each connecting path segment generates a dependency in the cooperation model. This is exemplified in the meeting scheduling example in the following section.

4.4.2 Meeting scheduler example continued

Figure 6 continues the example of scheduling a meeting. The dependencies are derived from the business workflow model in Figure 3 where there are three paths that connect the actor initiator

and the actor participant. The middle path segment is determined to be a resource dependency, because the participant is dependent on the initiator to provide preference and exclusion sets for a potential meeting. The upper path segment constitutes a state dependency indicating a requirement for directing the status of the meeting to be scheduled. The lower path segment indicates an interaction dependency, because the initiator is dependent on the participant to be informed on the most appropriate meeting dates.

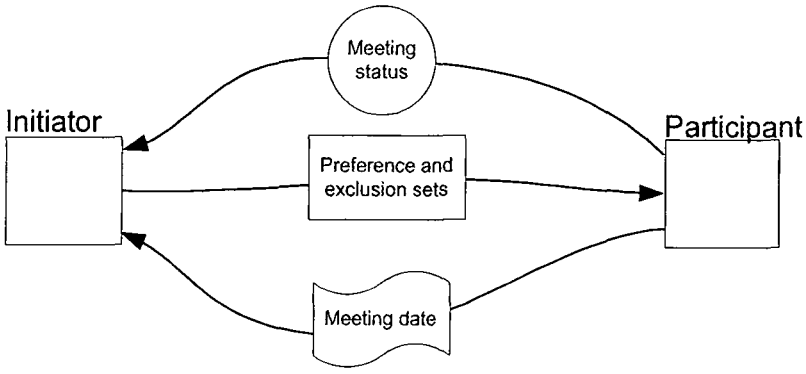


Figure 6: Cooperation model for scheduling a meeting

4.5 Articulation model

The purpose of the *Articulation model* is to identify what protocols need to exist for actors to cooperate with each other. The articulation

model is derived from the actor model and the cooperation model. The model is described in a tabular format (a table for each actor).

4.5.1 Articulation work

The notion of *articulation work* is a result of extensive studies in clinical work mainly carried out by the sociologists Anselm Strauss [Strauss 1986] and Elihu Gerson and Susan Leigh Star [Gerson and Star 1986]. Articulation work describes the number of secondary activities of coordinating and integrating cooperative structure and work processes. In other words, cooperating actors have to articulate (i.e. to divide, allocate, co-ordinate, schedule, mesh, interrelate, etc.) their individual activities: Who is doing what, where, when, how, etc.? With these significant issues, articulation work has recently been adapted as a foundation in CSCW research in particular by Kjeld Schmidt and Carla Simone [Schmidt and Simone 1996] and Geraldine Fitzpatrick [Fitzpatrick 1998] for the development of CSCW toolkits such as wOrlds [Fitzpatrick et al. 1996]. The concept of articulation work is described in Chapter 3.3.

4.5.2 Articulation protocols

Based on the notions of articulation work, this work introduces an articulation model into requirements engineering to identify what actors (human or software system) need to articulate with each other in order to achieve some goal.

Articulation protocols provide cooperating actors the context for their individual tasks and facilitate the group progress to achieve the work process goal. These articulation protocols may include monitoring of other activities, directing attention to other actors, assigning tasks, or handing over ownership.

The articulation model is derived from the actor model and the cooperation model.

When actors cooperate with each other in a particular business work situation, they articulate themselves. The identification of the relationship type helps in choosing the right articulation protocol. Each type of relationship has a set of predefined articulation protocols associated with it:

- Goal dependency: A goal dependency has an Achieve and a Maintain protocol type. For example, the Achieve protocol takes a goal name or a condition to be achieved as one of its parameters.

- Task dependency: A task dependency has a Perform, a Disapprove, an Accomplish, an Order, and a Reject protocol. The protocol parameters specify what task to perform.

- Resource dependency: A resource dependency is associated with a Provide, a Reserve, an Allocate, an Obtain, an a Locate protocol. The resource is provided when an actor makes a statement. For example, a provide protocol may contain entities of a complaint such as address.

- State dependency: A state dependency is associated with a Monitor or Direct attention protocol. The protocol parameters specify what state to monitor.

- Interaction dependency: An interaction dependency is associated with an Inform or a Request protocol. The protocol parameters specify what interaction to use.

The articulation protocols are summarised in Table 4.

ACTOR RELATIONSHIP	PROTOCOL TYPE	COMMENT
GOAL dependency	Achieve	Achieve a goal
	Maintain	Maintain a goal
TASK dependency	Perform	Perform a task
	Disapprove	Disapprove a task
	Accomplish	Accomplish a task
	Order	Order to do a task
	Reject	Reject to carry out a task
	Accept	Accept a task
RESOURCE dependency	Provide	Supplier provides resource to dependent (Information resources (such as documents or files), material, technical, infrastructure)
	Reserve	Reserve a reserve
	Allocate, locate	Allocate a resource
	Obtain/block access	Obtain access of a resource
	Locate	Locate a resource
STATE dependency	Monitor	Monitor state
	Direct attention	Direct attention to a particular state
INTERACTION dependency	Inform	Individual inform interaction ((direct attention or hand-over)
	Request	Individual request interaction (assign a task)

Table 4: Articulation protocols for the different relationships

The list of articulation protocols types is by no means a comprehensive one. New protocols may be added with different relations as needed.

4.5.3 Meeting scheduler example continued

In the cooperation model, the cooperation dependency between the initiator and the participants is identified as a resource dependency. A resource dependency has five types of articulation protocols associated with it. The tasks captured in the actor model, in conjunction with the cooperation dependency identification, help to construct the articulation protocol in Table 5.

PROTOCOL TYPE	MEDIUM	DATA OBJECT	COMMENT
Provide(preference and exclusion sets)	system(send email)	Name, date	initiate meeting

Table 5: Articulation model for scheduling a meeting

Here, only an example of discovering the articulation protocol for the resource dependency is shown. Articulation protocols for other dependencies can be discovered in the same way.

The one line of the articulation model shows what is being sent, defining the medium and data objects.

4.6 Summary

This chapter presents a derivation modelling method that allows the modelling, analysing and communicating of requirements for business workflow systems. It provides a means of both visualising the behaviour of actors and defining how cooperative behaviour can be achieved.

The method is split into four main models, which are required to analyse and model requirements for business workflow systems:

- The *Business workflow model* identifies actors and their behaviour.

- The *Actor model* describes the actors behavioural structure of actors discovered in the business workflow model.

- The *Cooperation model* describes actor relationships in terms of cooperation interdependencies.

- The *Articulation model* describes what articulation protocols need to exist for actors to cooperate with each other using a software system.

The novel aspect of this work can be stated as follows:

- It presents a scenario-based derivation modelling approach in which models are transformed through a series of modelling aspects involving coordination and cooperation which are addressed by using what are effectively extensions of current requirements engineering methods.

- It supports clear and structured views of cooperation properties, and allows the derivation of articulation protocols from business workflow models in a scenario-driven manner. This allows requirements engineering to define how the expectations of the cooperative situation are to be fulfilled by the system to be built.

These novel aspects attribute the statement of requirements engineering for business workflow system that reflects the richness of these systems and also acts as a feasible starting point for development.

The scheduling of a meeting example has been used to illustrate the application of the approach.

- The approach integrates some of the important results from sociological work in the CSCW research area with the field of requirements engineering. In particular, the method draws on results from the sociologists Anselm Strauss [Strauss 1986], and Elihu Gerson and Susan Leigh Star [Gerson and Star 1986].

Chapter 5 presents a real-world case study in which this method is applied to.

5 Case study: Complaint Management in a Bank

This chapter presents the application of the scenario-based derivation modelling method defined in Chapter 4 to a real-world case study example of complaint management in a bank. As a result, the derivation of articulation protocols from business workflow models define the expectations of the cooperative situation can be fulfilled by the system. Various subtleties were found during the case study, which suggest refinements to the method. The refined method is presented at the end of this chapter.

5.1 Introduction

One of the motivations for the method presented in Chapter 4 is the ability to deal with business workflow situations: to support humans in expressing and reasoning about cooperative behaviour, in search for an appropriate set of articulation protocols to define how the

expectations of the cooperative situation can be fulfilled by the system to be developed.

The method describes guidelines in terms of rules supporting the derivation modelling approach.

This chapter presents fragments of a real-world example. The example, complaint management in a bank, is used in this chapter to evaluate the method presented in the previous chapters. This approach is based on the concept of "industry-as-laboratory" [Potts 1994] research. The case study of complaint management was chosen to try out the proposed method in a real problem situation and to ensure that the problem reflects enough reality in terms of size and complexity.

Section 5.2 describes the case study approach and the method applied.

Section 5.3 presents an informal description of the complaint management problem.

Sections 5.3.1, 5.4.2, 5.4.3 and 5.4.4 give the samples of the derived models using the complaint management example. Section 5.4.5 presents a candidate workflow system architecture for the complaint management case based on the results of the derivation modelling approach.

Section 5.5 describes several lessons learnt, which helped to refine the derivation modelling approach presented in Section 5.6.

5.2 Case study approach

The approach taken for this case study is an iterative and incremental one. This is the common approach in early-phase requirements elicitation and modelling, since the knowledge for the system to be built is distributed among many stakeholders and nowhere recorded in a systematic written form.

However, the novel aspect of this work maps to the needs of requirements engineering for business workflow systems as described in Chapter 2.2. In particular, it encourages a derivation modelling approach in which the various models properties support a clear and structured view of the described concepts identified in Chapter 3.

In the first step, the business workflow model is constructed, using slightly modified Use Case Maps [Buhr 1998], which superimposes causal paths for scenarios on a structural substrate of actors and organisational entities. In this step, the model aims at defining operational aspects (e.g. tasks of actors) and macroscopic behaviour of cooperating actors with some specific purpose.

In the second step, the actor model is derived from the business workflow model. It describes the behavioural structure of the actors involved in the business workflow.

In the third step, the cooperation model describes the actor dependencies. The dependencies are derived from the coordination expressed in the business workflow model. Coordination is captured

in the model by path segments that connect two actors. In addition, the analysis of tasks may lead to the discovery of cooperation dependencies.

In the fourth step, the articulation model identifies what protocols are needed in order for the actors to cooperate with each other for the workflow. The articulation model is derived from the actor model and the cooperation model. The content of articulation protocols is determined by the goals that satisfy the expectations of the cooperative situation.

The practicality of the approach is demonstrated in the following sections by modelling and analysing the requirements for the complaint management problem of a bank. However, the lessons learnt are presented in Section 5.5.

The next section gives an informal description of the case study.

5.3 Complaint management in a bank

This section describes informally the current complaint management situation. The current situation is described in order to discover what is currently unsatisfactory, and dually what could be considered satisfactory. The description of the current situation is called an *indicative* description; and what the situation should be like is called an *optative* description [Jackson and Zave 1995]. The indicative business workflow model and the current system architecture are

described in the following. The optative business workflow model and the derivation modelling are described in order to derive a candidate business workflow system architecture based on articulation protocols.

5.3.1 Indicative business workflow model

The bank in question currently has got some complaint management in place. All complaints are considered as negative comments expressed by customers with the purpose to improve the commented issue.

The goals of the complaint management are to regain customer satisfaction and to increase customer loyalty, and to recognise weaknesses, which indicate to enhance processes.

Customers can express complaints through a variety of channels of the bank. The customer can place complaints either orally (i.e. by phone or personally) or written (e.g. letter, fax).

Figure 7 illustrates the current business workflow at the bank. The business workflow model explains how the complaint management workflow is carried out by actors. Interactions are not explicitly shown to prevent unnecessary cluttering of the diagram. Interactions are implicitly shown by the UCMs. In the model, the UCM scenario paths show the activities performed for a specific scenario.

All complaints are handled via branches. The complaint management workflow is hardly structured and predictive.

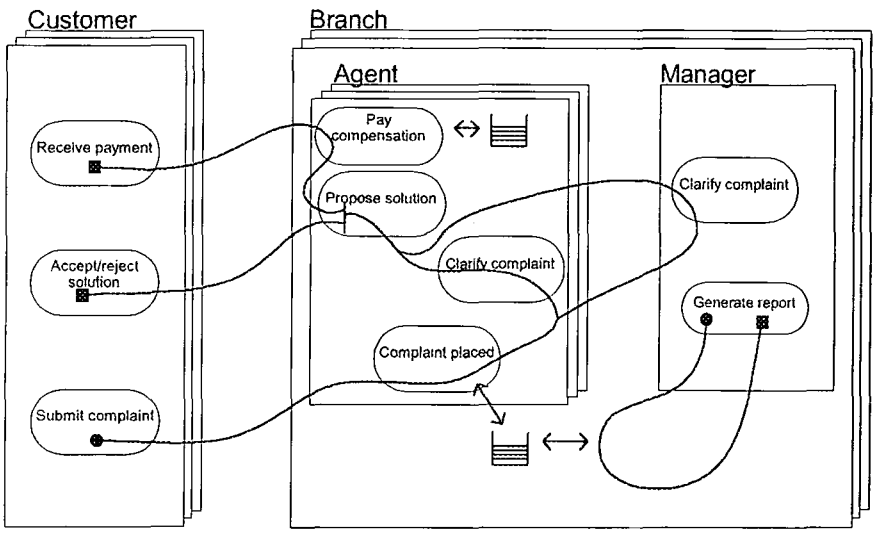


Figure 7: Indicative business workflow for complaint management

5.3.2 Indicative system architecture

The current system used to support complaint management is illustrated in Figure 8. Based on a simple database system, only actors and the supervisor are able the directly use the system. They are able to submit and query complaints. In addition, supervisors are able to generate reports.

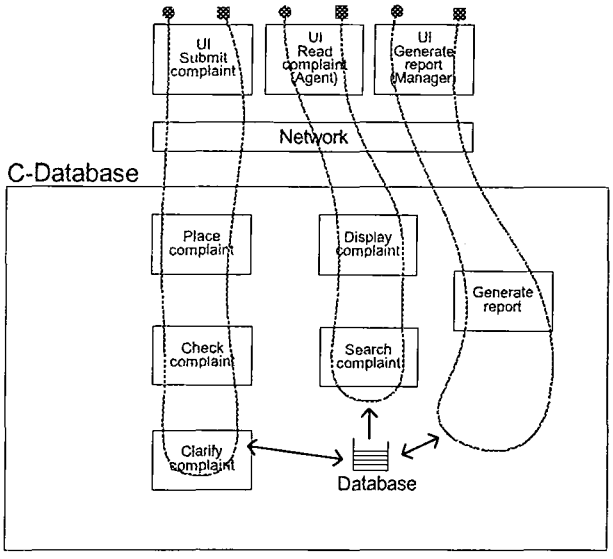


Figure 8: Indicative business workflow systems architecture

5.4 Optative complaint management model

The previous section gives a brief overview of the complaint management situation in the bank. Due to increasing competition in the market, the bank has felt the need to provide better services in terms of handling customer complaints.

In the following sections, the derivation modelling method is applied to derive articulation protocols from business workflow models, to define how the new situation can be satisfied by a workflow system.

5.4.1 Business workflow model

In this section, the revised business workflow model is presented. It explains how actors should do the business workflow in the future.

Several new basic principles for complaint management are essential for the bank:

- customers can choose the complaint communication channel (e.g. a branch or new Service Centre) and medium (e.g. personally or letter),
- all complaints are registered,
- as many complaints as possible are closed during the first contact,
- customers can be notified anytime about the status of their complaints, and
- systematic analysis of complaints is done for future improvement.

Based on these principles, particular interest in this example is the development and implementation of a complaint management workflow system in the bank. The system should enable the submission and processing of complaints as well as the ability to support workflow processes in order to transfer complaints between different organisational units. Employees of the bank register complaints without considering the medium how customers express

complaints. People of the organisational units are able to view complaint information from the system. Therefore, transparency of complaints increases in terms of its processing.

In addition to the new system, a new complaint Service Centre is introduced in the bank where customers are able to express complaints. The aim is to process as many complaints as possible at the Service Centre, most of them during the first contact, in order to shorten processing times and to decrease the processing effort for branches. Customers of the bank can contact the complaint Service Centre by a variety of communication media.

In general, complaint management in the bank is divided into three phases: entry, processing, and closure. During the placement of a complaint, the question of responsibility within the bank is clarified. Complaints are processed to the responsible agent and customers receive confirmation. In the second phase, complaints are processed. Solutions are developed and ideas about future enhancements of similar problems are proposed. After that, the customer is notified about the proposed problem solution and the complaint is closed, which may include the initialisation of payments.

The bank allows several channels for complaints (such as branch, service centre) and media (e.g. personal, telephone, e-mail, letter, fax). Customers can choose the media for placing their complaints at the bank. Confirmations of and solutions proposals for complaints may be communicated through these media, too.

Through the introduction of the new Service Centre into the bank-wide complaint management, branches forward complaints to the Service Centre agent.

The workflow system should support the basic principles and the phases described. Branches and Service Centre, which are involved in the complaint management, will use the system in some respect.

Figure 9 presents the business workflow model for complaint management as is defined for the future.

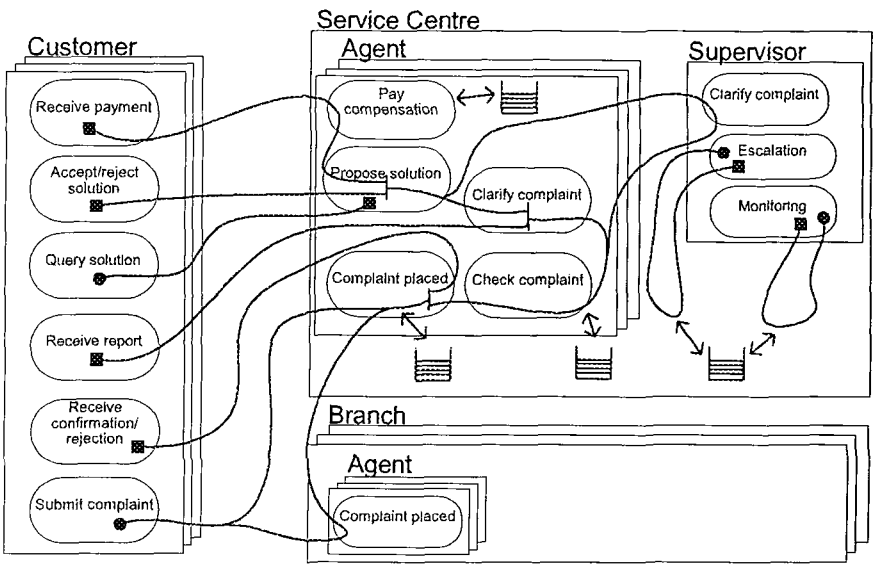


Figure 9: Optative business workflow model for complaint management

5.4.2 Actor model

The actor model describes the behavioural structure of the actors discovered in the business workflow model. Actors are described in

terms of their tasks. Initially, it was suggested in Chapter 4.3 to describe be behaviour of actor in terms and goals. However, during the case study it turned out that the distinction between goals and tasks is a very difficult one to make, though the business workflow model was easier to understand. It was decided to describe actors only in terms of their tasks, preconditions, and postconditions.

The mapping from the business workflow model to actor model elements has been done as follows: In order to derive the actor models for the actors (customer, agent of branch, agent of Service and supervisor of Service Centre) involved in the complaint management workflow, the different UCM path segments that cross them were examined. From the business workflow model (Figure 9), each of the tasks identified is inserted into the actor model table. The preconditions and postconditions are captured in the corresponding row. The actor models are shown in Table 6 (Customer), Table 7 (Agent of service centre), Table 8 (Supervisor of service centre), and Table 9 (Agent of branch). Preconditions and postconditions are omitted partially and are only exemplified in certain tasks.

TASK	PRECONDITION	POSTCONDITION	COMMENT
Submit complaint	Customer has valid account no.	Complaint expressed	
Receive confirmation	Complaint placed	Customer notified	
Receive rejection			
Receive report			
Query solution			
Accept solution			
Reject solution			
Receive payment			

Table 6: Actor model for customer

TASK	PRECONDITION	POSTCONDITION	COMMENT
Place complaint			
Check complaint			
Clarify complaint			
Propose solution			
Pay compensation			

Table 7: Actor model for agent of service centre

TASK	PRECONDITION	POSTCONDITION	COMMENT
Clarify complaint			
Receive escalation			
Monitor complaint			

Table 8: Actor model for supervisor of service centre

TASK	PRECONDITION	POSTCONDITION	COMMENT
Place complaint	Complaint submitted	Complaint placed	

Table 9: Actor model for agent of branch

5.4.3 Cooperation model

The cooperation model describes the actor relationships in the complaint management workflow. The relationships are derived from the cooperation and tasks expressed in the business workflow model. Cooperation is captured in the business workflow model by scenario path segments that connect two actors in the complaint management workflow, as described in Chapter 4.4.

The cooperation model relates an actor that provides a service to an actor who requires that service in the complaint management workflow. Figure 10 shows the cooperation model on the basis of the business workflow model presented in Figure 9. Each scenarios path segment in the Use Case Map that connects two actors generated a dependency in the cooperation model. For example, the dependencies between the actor Customer and the actors Agent (Service Centre) and Agent (Branch) have each determined a goal dependency called 'submit complaint'. The agents are dependent on the customer to place a complaint at the bank, if the customer is not satisfied with a particular circumstance.

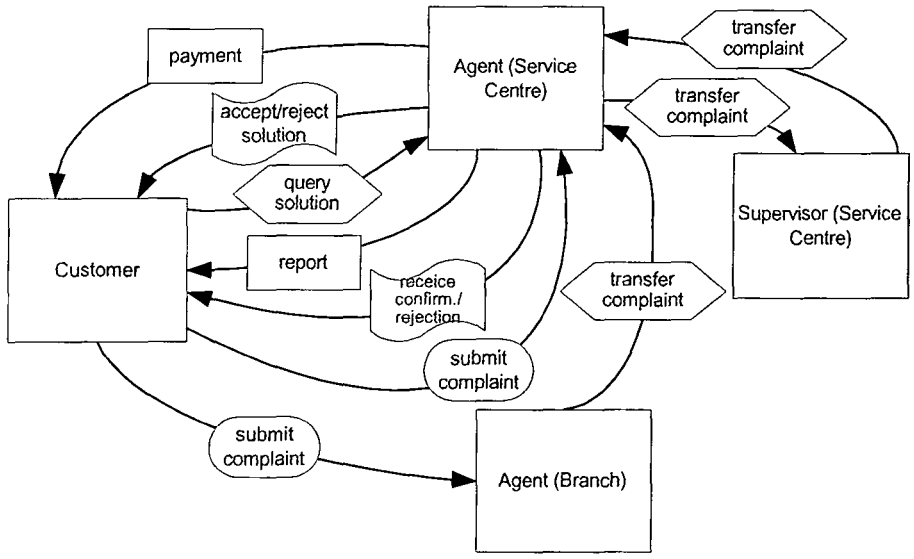


Figure 10: Cooperation model for complaint management

An interesting dependency can be found with the collaboration dependency. A collaboration dependency, as for example in 'accept/reject solution', constitutes a loop indicating a requirements for negotiation. The Agent (Service Centre) and the Customer may

need to collaborate and negotiate to determine a final and acceptable resolution for the particular problem. This observation is in vein with Schäl's [Schäl 1996] definition of collaboration, where collaboration requires actors to work together to achieve a common goal, under the condition that a contribution is needed by each participating actor.

Collaboration dependencies can be considered as one of the difficult situations within workflow management. Such dependencies cannot be simply seen as one-way interaction, but as situations where actors need to articulate through protocols. This is described in the next section.

5.4.4 Articulation model

As describe in Chapter 4.5, the purpose of the articulation model is to identify what protocols need to exist for actors to cooperate with each other. The articulation model is derived from the actor model and the cooperation model. The articulation model basically defines what protocols need to exist to fulfil the dependencies identified in the cooperation model. The content of such articulation protocols is determined by tasks that satisfy the dependencies, which are captured in the actor model.

During the derivation of the articulation model in the course of the case study using the steps described in Chapter 4.5, various issues were found that led to a refinement of the articulation model.

The revised approach for the articulation model consists of two steps. First, a cooperative interaction model identifies the interactions needed for the actors to cooperate with each other. Second, a articulation protocol specifies the services provided to each other. It defines the expectations of how actors can fulfil identified cooperation dependencies.

Cooperative interactions are described in tabular form – a table for each actor. The table has three columns: *actor*, *receive*, and *do*. The *actor* column contains the actor that receives and does the interaction. The *receive* column defines the interactions received by the actor. The *do* column defines all possible responses to each received interaction.

Table 10 shows the protocol types defined for the case study. The collaboration dependency resulted into four protocol types: Propose, Re-propose, Accept, and Reject. These four protocol types together can be seen as a general collaboration mechanism between actors to achieve a common goal.

COOPERATION RELATIONSHIP	PROTOCOL TYPE	DESCRIPTION
GOAL dependency	Achieve	Achieve a goal
TASK dependency	Perform	Perform a task
RESOURCE dependency	Provide	Supplier provides resource to dependent (Information resources (such as documents or files), material, technical, infrastructure)
	Request	Reserve a resource
COLLABORATION dependency	Propose	Propose an issue
	Re-propose	Re-propose the issue
	Accept	Accept proposal
	Reject	Reject proposal

Table 10: Articulation protocol types for relationships in the case study

For example, a collaboration dependency was identified for the proposal of a problem solution to the customer. The agent proposes a solution and the customer may respond. The agent may get from the customer in return a modified solution (often some sort of payment), or the customer accepts or rejects the solution completely. If the agent receives a re-proposal, then the agent in return needs to evaluate the proposal and get back a proposal again to the customer (see Table 11).

ACTOR	RECEIVE	DO
Agent		Propose solution
Customer	Propose solution	Accept or reject solution
Customer		Re-propose solution
Agent	Re-propose solution	

Table 11: Cooperative interaction between customer and agent (service centre)

One major result of the case showed that articulation protocols should not only be considered as simple interactions that must occur in order for actors to cooperate. Articulation protocols define some sort of a contract that is established between actors in terms of services provided to each other.

The purpose of articulation protocol is to define expectations of how actors can fulfil cooperation dependencies as well as the tasks they have defined by the actor model. Cooperative interactions are used as guidelines for discovering those expectations.

An articulation protocol consists of four parts: *actors*, *permitted services*, *guaranteed services*, and *rules of service*. The actors list the actors involved in the articulation protocol. The *permitted*

services section specifies the services that actors can make available for each other. The *guaranteed services* section specifies which services an actor must provide or use.

It is simple to decide if a service is a permitted or a guaranteed service. A service that is required by an actor becomes a guaranteed service by the other actor. A service owned by an actor belongs to the guaranteed service section if the actor permits or requires the other actor to use that service. For example, the bank provides to their customers an e-mail service as a guaranteed service for submitting their complaints. For some reason, the bank decides to not continue providing this service.

The *rules of service* section specifies quality and capacity of services and information on their usage. Rules of services are either mandatory or desirable. For example, a customer always needs to provide a customer identification number when submitting a complaint, otherwise the complaint cannot be place at the bank. It is always desirable to provide the customer with precise information on the status of the complaint placed.

In the following, the way of building an articulation protocol is provided. The examination of the different cooperation dependencies captured in the cooperation model and the decision whether an articulation protocol is needed to capture the cooperation. If an articulation protocol is needed, it needs to be decided what permission, guarantees, and rules for the protocol have to be captured in the protocol.

Table 12 illustrates an example of an articulation protocol. The protocol is between the customer and the agent of the service centre. In this protocol, the agent of the service centre guarantees the customer to provide a solution proposal. The agent guarantees to use a personal communication channel if the complaint was placed personally (e.g. phone) or a written channel, respectively. The customer has the permitted services to receive a solution proposal. The rules of service clause states that if the customer cannot be reached within 48 hours to propose a solution for the problem personally, then the solution has to be provided to the customer in written form.

CUSTOMER	AGENT (SERVICE CENTRE)
Permitted services	
Solution proposal Report medium change	
Guaranteed services	
	Provide: solution proposal Use: personal channel if personal submission Use: written channel if written submission
Rules of service	
	Mandatory: report medium change: instead of personal proposal, written solution proposal if customer cannot be reached personally within 48 hours

Table 12: Articulation protocol between customer and agent (service centre)

5.4.5 Business workflow system architecture

The previous sections describe systematic derivation modelling approach. The method captures effectively the complexity of

business workflow problems, actor structure, cooperation dependencies, and articulation protocols. The practicality of the approach was confirmed by the application of the method to a real-world case study.

In this section, an architectural solution of the business workflow system is presented (Figure 11). It is based on the results derived from the derivation modelling approach, starting with the business workflow model and deriving finally articulation protocols, which provide detailed information of what a systems to be proposed needs to provide. The system architecture is presented using Use Case Maps. The business workflow system architecture is based on a 3-tier architectural style in which the system is decomposed into three major components: database, business functionality, and user interface. The division reflects the principle of separation of concern: a component should be responsible for one task only. Following this principle minimises the impact of change of one component on other ones. Furthermore, a 3-tier architectural style stands for distributedness and scalability, both of which are important quality attributes in business workflow systems.

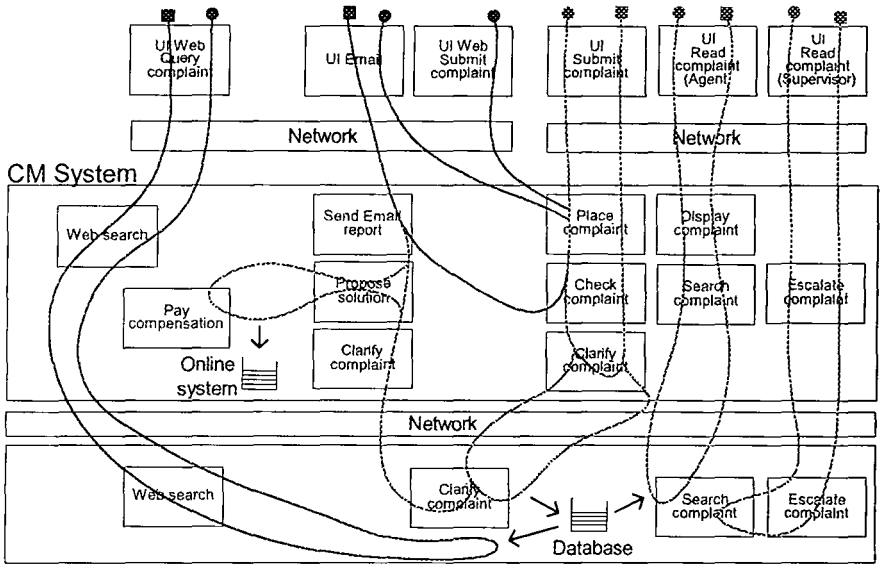


Figure 11: Optative business workflow system architecture

5.5 Lessons learnt

The above sections demonstrate that the method provided in Chapter 4 can be used for modelling and analysing real-world business workflow problems. The case study yielded various lessons, which are described in the following.

5.5.1 Separation of tasks

It was found valuable in the case study to separate tasks in the business workflow model to avoid cluttering in UCMs by expressing them separately. Previously, when employing it at the meeting scheduler example, no clear distinction was made. During this case study, however, it was chosen to express these tasks as follows: all

tasks (those that express actors' tasks) where included separately with rounded rectangles. For example, the tasks 'query solution' and 'accept/reject solution' may be seen as one task, i.e. providing a solution to the customer. However, the distinction is a remarkable one, since the initiation of the scenarios is on the different actors involved. Thus, it proved to be a very valuable thing to do.

This separation of tasks from responsibilities helped to make the business workflow model using UCMs more comprehensible and more precise for deriving the actor and cooperation model. The resulting business workflow model affords the ability to construct a more comprehensible model, while facilitating the construction of more complete and realistic derived models.

5.5.2 Collaboration dependencies

At times it was found beneficial to include a collaboration dependency in the cooperation model. For example, the dependency 'accept/reject solution' is a situation in which collaboration between actors may occur. The dependencies were refined in the sense that the collaboration dependency was introduced by using a generic protocol type: Propose, Re-propose, Accept, Reject. The inclusion of this collaboration dependency ensured that such collaboration situations are explicitly considered in the articulation protocols.

5.5.3 Distinction between cooperative interaction and articulation protocols

According to Jackson, separating concerns is simply a matter of structuring a complex topic as various more simple topics that can be considered separately [Jackson 1995]. In this case study, it was found that the original kinds of articulation protocols were not sufficient to express how cooperation work can be achieved between actors using a workflow system. It was clear that a better way to distinguish between the cooperative interaction and the services the needs to be provided must be identified. It was done during the course of this case study by the following:

Cooperative interactions express interactions that must occur in order to cooperate with each other. Depending on the dependency type, protocols provide generic mechanisms for the interaction.

The articulation protocol is to define expectations of how actors can fulfil cooperation dependencies as well as the tasks they have defined by the actor model.

This distinction between the cooperative interaction and the articulation protocol offers a clear separation of concerns for the interaction that must occur and by the services it is achieved.

5.6 Refined derivation modelling method

This section presents the refined derivation modelling method.

Five types of models are used within this approach during the requirements engineering process (Figure 12):

- The *Business workflow model* identifies actors and their behaviour. It gives a high-level view of the actors and workflows, and provides a starting point for deriving the details of the other models. It is generated by tracing workflow scenarios that describe tasks, actors, and their behaviour along the way.

- The *Actor model* describes the behavioural structure of the actors discovered in the business workflow model. The actor model is derived from the high-level workflow model and is described in terms of their goals and tasks.

- The *Cooperation model* describes actor relationships in terms of cooperation dependencies.

- The *Cooperative interaction model* describes the interactions needed for the actors to cooperate with each other.

- The *Articulation model* specifies the services provided to each other. It defines the expectations of how actors can fulfil identified cooperation dependencies. Identified cooperative interactions are used as guidelines for discovering those expectations. Articulation protocols consist of four parts: *actors*, *permitted services*, *guaranteed services*, and *rules of service*.

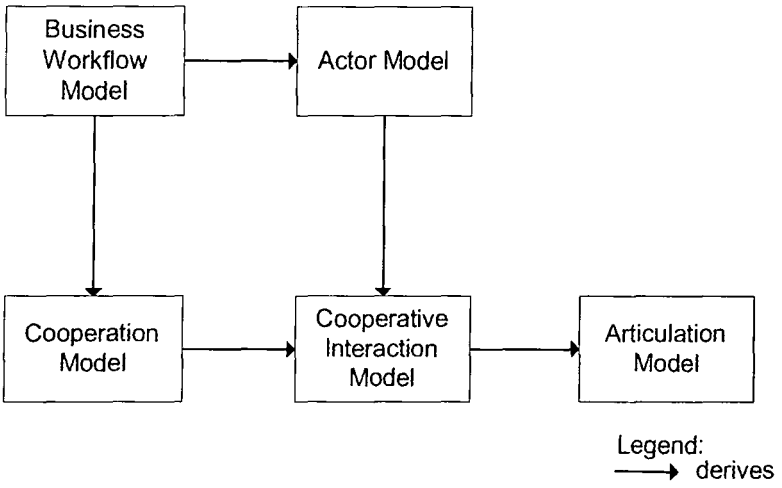


Figure 12: Refined derivation modelling method

5.7 Summary

The previous two chapters introduce a method, which was developed for modelling and eliciting requirements for workflow business systems. The method is based on a derivation modelling approach, in which articulation protocols are derived from business workflow models through defined steps to be carried out.

The case study presented in this chapter has been carried out by working through real-world examples how the method can be used to provide scenario-based support in requirements engineering. Some lessons learnt were found during the case study, which led to the refined derivation modelling method presented in Section 5.6.

6 Heuristics for Analysis and Construction

This chapter describes heuristics for analysis and construction used by requirements engineering practitioners applying the derivation modelling method. These heuristics are a set of rules, which guide the requirements engineering practitioner towards higher rate of success for analysing the available information and for constructing the models. They were derived from experiences and observations in applying the method in both the initial meeting scheduler example and the complaint management case study.

6.1 Introduction

The objective of this chapter is to describe some typical heuristics used by requirements engineering practitioners applying the method described in this thesis. In this thesis, heuristics are sets of rules, which guide the requirements engineering practitioner towards a higher rate of success. This method provides heuristics for the

identification of requirements and for construction for each of the models. The heuristics identified are described around requirements engineering for business workflow systems.

The lessons learnt in applying the meeting scheduler problem served as the origin for the ideas to formulate this method. The heuristics of this chapter were derived from these experiences and from experiences and observations made during the real-world case of complaint management, which was described in the previous chapter.

Section 6.2 presents a set of questions for analysis to support requirements engineers in applying the method. Since the utilisation of the heuristics depends upon the particular model with which the requirements engineer is involved at any given time, the sets of heuristics (6.3, 6.4, 6.5, and 6.6) are presented according to the four models described, with discussion of the application of the heuristics to specific activities.

6.2 Types of questions for analysis

This work offers a set of recurring questions, which follow the inquiry cycle approach [Potts et al. 1994] instantiated for requirements analysis. They have been adjusted to comprehensively explore and guide workflow situations in this work. This section discusses these types of questions; subsequent sections in this chapter suggest appropriate guidelines based upon the answers derived from the

questions asked. The types of questions are summarised below and discussed throughout this chapter:

- What-is: This question requests specific information regarding terminology, which is unclear to some stakeholder with no knowledge of the application domain.

- Who-is: This question requests specific information of the actor responsible for the given task or workflow.

- Why: This question requests reasons, which underlie work activities. For example, "Why is this information routed?"

- What-kinds-of: This question requests further refinements on some concepts. For example, "What kind of complaints should be supported?"

- What-if: This question may be asked if requirements engineers may try to explore a situation further in which an unexpected action might occur. An example may be: "What happens if a complaint cannot be solved at a branch?" These questions lead to the consideration of other actor and workflows that would be affected.

- How-to: This question requests some information how some action is performed.

- When: This question requests timing constraints for a given event. For example, "When is a customer complaint to be escalated?"

- Relationship: This question asks how one actor is related to another or how one task is related to another so that interaction can be established. For example, requirements engineers may consider each task and ask: "What tasks are prerequisites for this task?", "What tasks must follow this task?", and "What actor depends on this goal for completion of their task?"

These question types support requirements engineers in knowing when and how to apply the heuristics by providing guidance as to how much detail is needed before one can be reasonably confident that the user requirements are fully elaborated.

The following sections distinguish between two kinds of heuristics:

- Heuristics for analysis (HA): this type of heuristic provides rules and guidelines for identifying and analysing requirements of the business problem at hand.

- Heuristics for construction (HC): this type of heuristic provides rules and guidelines for constructing the various models defined in the previous Chapter.

Both kinds of heuristics are numbered sequentially throughout this chapter.

6.3 Heuristics for the business workflow model

This section presents heuristics for the business workflow model. They are based around the identification of tasks, actors, preconditions and postconditions, and constraints.

The elicitation and analysis is the process of exploring documentation, from information about the organisation to system-specific information, and discussions with stakeholders.

Scenario analysis heuristics support requirements engineers in uncovering concrete situations and circumstances. This section presents the heuristics to guide scenario analysis and construction.

(HA1) *An effective way to identify candidate scenarios for construction is to ask: What are the circumstances under which this workflow can occur? The identified scenarios are elaborated by listing the activities that must occur should the scenario take place.*

(HA2) *The scenarios which requirements engineers should pay particular attention to are those which violate others.*

(HC1) *A scenario is represented as Use Case Map.*

Example 1: Consider Figure 3 and Figure 4 for illustration for a scenario and decomposition of the scenario.

In business workflow systems requirements, the identification of tasks is one of the crucial issues. Tasks may be identified before or after the identification of actors.

(HA3) *Tasks are named in a standardised subset of natural language in which the first word is a verb that describes the kind of task being named.*

(HC2) *Tasks can be modelled as responsibilities in UCM terminology. A responsibility may consist of more than one task.*

(HC3) *Stubs allow the hierarchical decomposition of complex UCMs. Stubs enable to draw UCMs that give a high-level overview of the general trend of paths, while leaving details that might obscure the big picture to sub-UCMs shown in separate UCM diagrams.*

(HA4) *Actions words that point out some state that is or can be achieved once the task is completed are candidates for the system. They are identified by asking: Does this behaviour or task denote an action to be performed?*

The task heuristics offer requirements engineers approaches to identify tasks, based on the available information. The actor heuristics allow requirements engineers to identify and analyse actors. They are discussed in the next section.

Actor analysis heuristics support requirements engineers in identifying and analysing actors who are responsible for particular tasks. Every task has at least one responsible actor, be it a person, organisation, or even a system. This section presents the heuristics to guide actor identification and task allocation.

(HA5) *At least one actor must be responsible for a task. If it is not possible to allocate responsibility for a task, then it can be assumed that the task lies beyond the scope of the proposed system.*

(HA6) *Actors may be identified by considering each task and asking: Who or what actor could be responsible for that task? The answer to this question will be the name of the responsible actor.*

(HA7) *Actors may be a human, an organisation, or the system.*

Example 2: In a complaint management system, the task escalate complaint is the responsibility of the system. Depending on the desired implementation, the actor may be either the workflow system or a human actor.

(HC4) *Actors are modelled as components in UCMs.*

(HA8) *Different actors may be responsible for the completion of a task at different times.*

(HA9) Multiple actors may be associated with one task.

(HC5) Multiple actors are represented by stack in UCMs. Stacks imply that each actor is distinct but operationally identical from the perspective of the traversed path.

Example 3: Consider Figure 9 of the complaint management example. Service Centre agents are modelled as stack, because they may be more than one agent at a time.

The actor heuristics offer requirements engineers approaches to identify and classify actors. The pre- and postcondition heuristics allow requirements engineers to identify and analyse conditions. They are discussed in the next section.

Pre- and postconditions place some constraint on the achievement of a scenario or task. Pre- and postcondition heuristics support the requirements engineer in identifying pre- and postconditions for scenarios. These heuristics are presented in this section.

(HA10) Each scenario has a precondition and one or more postconditions.

(HA11) Pre- and postconditions can be identified by considering each scenario and asking: What condition is imposed on the scenario?

(HA12) Preconditions can be identified by searching for temporal connectives (i.e. during, before, after, etc.). When a scenario can be completed, then the precondition can be become true.

Example 4: Chapter Figure 9 illustrates how the identification of the temporal connective *before* led to the identification of the precondition: complaint at branch can not start, before a customer complains at a branch.

(HA13) Postconditions can be identified by the required result of a scenario.

Example 5: In the meeting scheduler example in Figure 3, this scenario has two postconditions. First, the initiator is informed on whom attends the meeting, and second, the participants know the meeting date.

6.4 Heuristics for the actor model

This section presents heuristics for the actor model. Heuristics for the actor model allow the requirements engineer to derive the actor model from the high-level workflow model. The heuristics in this section support in mapping path segments from high-level workflow models to goals in the actor model.

(HC6) *Path segments that traverse an actor represent goals to be achieved by an actor.*

Example 6: Table 2 in Chapter 4.3.3 illustrates how the path segments *Initiate meeting* and *Inform on status* in the meeting scheduler example are represented as goals to be achieved by the meeting initiator.

(HC7) *Scenarios may share some common path segment.*

(HC8) *Stubs represent sets of tasks. If a path segment has responsibilities or more than one stub, then the path segment should be mapped to a goal in the actor model.*

(HC9) *Pre- and postconditions map to pre- and postconditions. Preconditions must hold in order for goals or tasks to be performed. Postconditions are the effects of performing a successful goal or task.*

Example 7: Consider for example Table 6 of Chapter 5 in the complaint management problem. The precondition *Complaint initiated* and the postcondition *Complaint assessed* are allocated to goal *Problem evaluated*.

(HC10) *Responsibilities along a path constitute task.*

6.5 Heuristics for the cooperation model

This section presents heuristics for the cooperation model. Heuristics for the cooperation model allow the requirements engineer to derive the cooperation model from the high-level workflow model.

(HA14) The cooperation model identifies five types of actor dependencies: goal, task, resource, state, and interaction dependencies.

(HC11) Each path segment that connects two actors in the high-level workflow model derives a dependency in the cooperation model.

Example 8: Figure 6 in the scheduling a meeting problem illustrates how the dependencies are derived from the high-level workflow model.

The heuristics for the cooperation model offer requirements engineers approaches to derive the dependencies of the cooperation model, based on the high-level workflow model. The heuristics for the articulation model allow requirements engineers to derive the articulation model from the actor model and the cooperation model. They are discussed in the next section.

6.6 Heuristics for the articulation model

This section presents heuristics for the articulation model. Heuristics for the articulation model allow the requirements engineer to derive the articulation model from the actor model and the cooperation model.

(HA15) Each type of actor relationship has a set of predefined articulation protocols associated with it.

Example 9: Table 4 shows the protocols types for all actor relationships.

(HC12) The articulation model is described in tabular form, using one table for each actor.

(HC13) An articulation model consists of a set of articulation protocols, which describe services provided to each other. It defines the expectations of how actors can fulfil identified cooperation dependencies. Identified cooperative interactions are used as guidelines for discovering those expectations. Articulation protocols consist of four parts: actors, permitted services, guaranteed services, *and* rules of service.

6.7 Summary

The chapter presents a set of heuristics for analysis and construction, which support the requirements engineering practitioner in applying an inquiry-based approach. The heuristics detailed in this chapter are:

- Heuristics for the high-level workflow model
- Heuristics for the actor model
- Heuristics for the cooperation model
- Heuristics for the articulation model

The following chapter discusses the evaluation of this method.

7 Evaluation

This chapter presents the evaluation of this work. It describes the principal ideas behind the method and indicates major problems found while developing the method and justifying this approach. It reiterates the properties that such a method should display; it then evaluates the work by discussing its weaknesses and strengths with regard to these properties. Finally, this chapter examines the suitability of the visual scenario-based technique and discusses experiences while applying the method to case studies.

7.1 Design rationale

This section discusses briefly the history of this research, focusing on the most significant problems that were addressed before it reached the final stage presented in Chapter 5.6.

7.1.1 History

The development of the method mainly benefited from being applied to different problems of business workflow systems and regular literature surveys.

Initially, the starting point of the research was to take a small number of concepts that seemed promising for describing and understanding business work context. The concepts provided a general way of describing dependencies between actors and the various mismatches that might exist. The concepts were applied to a case study of collaborative authoring among scientists and engineers of a multi-national chemical company [Strassl 1996].

Some concepts for 'intention' and 'behaviour' were subsequently provided, implementing a distinction made most famous by Lucy Suchman between 'plan' and 'action' [Suchman 1987]. The insight here was that it is both possible and useful to relate these models of dependencies to models of behavioural interaction between the proposed system and the actors who are playing roles identified in the intentional dependencies.

Building on the work on intention, simple graphical concepts for modelling commitments and expectations between cooperating actors were used [Strassl and Smith 1997] until the appearance of Buhr's Use Case Maps [Buhr and Casselman 1995] [Buhr 1998], which influenced the method and helped considerably in developing it to its current form.

Later, the particular issue in cooperative settings – articulation work – was addressed to account for work being done about work. The fundamental distinction was originally introduced by the sociologist Anselm Strauss [Strauss 1986] in order to account for the observation that cooperating actors need to articulate their individual activities by considering who is doing what, where, when, how, by means of which, and under which constraints [Strassl and Smith 1998].

In summary, the method, as described in this thesis, is a result of successive refinements and the effort to encapsulate cooperative work aspects into this method.

This chapter explains the problems that were faced during the development of the method. It discusses the decisions made and explains why some solutions were rejected in favour of others. The next section discusses the most significant problems that were to solve, before it reached the stage presented in Chapter 5.6.

7.1.2 Major problems and their resolutions

This section discusses the most significant problems to be solved in order to develop the method:

- Set of models: A significant decision was to define the set of models to be used in the method. The first solution in the collaborative authoring design study made use of an intentional

model and a dependency model. The dependencies between the actors involved were described using concepts of commitment and expectation. For example, a role may have a commitment to do something for other roles or provide something for it. In the collaborative authoring example, a coordinator commits to defining the document structure, and to setting reasonable deadlines for authors and reviewer to submit their contributions. However, this did not sufficiently describe all aspects needed. It was necessary to define further models, using aspects of cooperative work – such as collaboration and articulation. In this way, the richness of business workflow situations can be adequately reflected.

- Articulation protocols: An articulation protocol serves to define expectations of how actors can fulfil their cooperation dependencies and so to achieve some task or goal. The first solution was to define the protocol type, a medium by which the task may be achieved, and the data objects used. However, it was discovered that this solution is unsatisfactory, because it did not provide enough information on what kind of services are permitted or guaranteed between the cooperating actors, and so problems may occur later in the development of the system. For this reason, articulation protocols consists of permitted services, guaranteed services, and rules for these services.

- Derivation modelling: After having decided on the various aspects to be described in the models, a method was required to derive articulation protocols from other models, allowing a requirements

engineer to define how cooperative business workflow situations are to be fulfilled by a system. Given that the models were represented either in graphical from (e.g. Use Case Maps for the business workflow model) or in tabular from (e.g. for articulation protocols), it turned out to be one of the difficult problems that was to be solved. The introduction of derivation rules and the application of the derivation modelling method to case studies helped to provide pragmatic heuristics and guidelines that support real-world requirements engineering practitioners.

- Role of scenarios: After having defined business workflows and cooperative work models, it turned out to be of high value in talking to stakeholders to be able to show alternatives and exception handling situations. Initial graphical models for commitments and expectations offered only limited help in modelling scenarios, because it had been necessary to define the use of scenario from scratch, which was not the focus of research. The appearance of Use Case Maps influenced this work, since they offered the possibilities of represent scenarios as hereby to intertwine behaviour and structure. Soon the role of scenarios expanded since they helped enormously in understanding the expected dynamic behaviour of the system, and in identifying articulation protocols between actors during the requirements engineering phase.

- Cooperation dependencies: Initially, cooperation dependencies between actors were described using concepts such as commitments and exceptions that should satisfy behavioural

models of such situations. It came to light that this approach is not sufficient. Instead, it was decided to use cooperation dependencies similar to those introduced in Yu's Strategic Dependency Model, which provides a variety of dependency relationships among actors [Yu and Mylopoulos 1994] [Yu 1995]. However, these types of dependencies were not enough for our purposes, as highlighted in the case studies. Dependency types such as state and interaction were integrated into the cooperation model. Finally, during the complaint management case the cooperation dependencies were refined again, because the need to express and reason about negotiation was found in numerous situations. The collaboration dependency in the cooperation model constitutes a generic mechanism for negotiation for cooperating actors to achieve one common result, whereby a contribution is needed by all parties involved.

7.1.3 Summary

The previous sections describe the progression in the ideas developed in this work. Major problems found while developing this method, justifying the approach, were presented. It discusses the most significant problems that were to be solved in the development of the derivation modelling method. These are:

- the decision on the appropriate set of models,

- the definition of the articulation protocol,
- the introduction of derivation rules and heuristics,
- the integration of scenario modelling techniques, and
- the definition of cooperation dependencies to express and to reason about negotiation.

During this discussion, the temporary solutions adopted were described along with the reasons that led to rejection. The final method is presented in Chapter 5.6.

7.2 Assessment

The method presented in Chapter 5.6 is the final method of this work while investigating the advantages of a scenario-based derivation modelling approach to requirements engineering for business workflow systems. Existing methods suffer from two fundamental problems, as explored in Chapter 2:

- the lack of modelling concepts with cooperation aspects, and
- the lack of methodological guidelines.

A scenario-based derivation modelling method should exhibit a number of properties. In subsequent sections, the following properties are analysed and assessed:

- the ability to derive articulation protocols from business workflow models through scenario-based derivation modelling,
- the ability to use articulation protocols as a starting point for development, and
- the ability to use Use Case Maps as the visual modelling notation for these purposes.

Finally, the main findings made are presented from observations made during the application of this method to several case studies.

7.2.1 Scenario-based approach

Business workflow system implementations require deep understanding of business and human cooperation. This work is concerned with enhancing part of the requirements engineering process for such systems.

Chapter 3 proposes that the use of a derivation modelling approach that is based on a scenario modelling technique, such as Use Case Maps, during the requirements engineering process can overcome many deficiencies. As a matter of fact, Use Case Maps are good at

describing, in a high-level way, how the organisational structure and the emergent behaviour of complex business workflow situations are intertwined. Moreover, the notation is not a behavioural specification technique in the ordinary sense, but a notation for helping humans to visualise, think about, and explain the big picture. It represents causal scenario paths as a set of lines threading through components without the scenarios actually being specified in a too detailed manner. The composition of scenario paths can be easily described as visual behaviour structures. Since paths are continuous and notational lightweight, many workflow paths may be combined in a single map in a way that enables "the mind's eye" both to see them together and to distinguish them.

This work considers scenarios as an *engine for design* [Mack 1995] during requirements modelling for business workflow system to stimulate, facilitate and document shared understanding between stakeholders – its occurrences, assumptions, action opportunities and risks.

7.2.2 Derivation modelling

It would be good to be able to reason formally about requirements of a new business workflow system as soon as possible in the development process. The advantage gained would be the ability to show that a specification met the requirements and maybe use prototyping to refine the requirements and to correct errors,

ambiguities and inconsistencies early. However, it is impossible to develop an approach that formalises such a process.

In reality, the requirements engineering phase starts from a set of highly informal requirements and may include the capture of the requirements, involving extensive discussions with stakeholders of the future system. Thus, an analysis method cannot provide a formal process, as it is impossible in practice to expect stakeholders being mathematicians, able to express their needs and goals by means of, for example, a set of equations. The point of the 'formal' and the 'informal' is also well observed by Joseph Goguen [Goguen 1992] [Goguen 1996].

Nevertheless, the derivation of articulation protocols from business workflows reduces the distance between highly informal and incomplete requirements models and more rigorous methods for designing systems by providing a method which reflects earlier the richness of cooperative work properties as usual. Shortening this distance is one of the main goals of this work, and a primary result of the derivation modelling method. The final output provided by the derivation modelling method is a set of articulation protocols which can then be used as the starting point of a more formal design process, as mentioned in Section 7.1.2.

In order to allow derivation, the method has the following characteristics:

- It produces a set of articulation protocols, which includes important properties that reflect cooperative work settings.

- It proposes methodological steps to be followed when constructing the intermediate models and final models (the business workflow model is based on Use Case Maps; the actor model and the cooperation model can be produced from information in the business workflow model; the articulation model can be created by following the heuristics).

- It provides a development process, by offering a set of derivation rules, heuristics, and mappings from the business workflow model to articulation protocols.

Therefore, this work is not a rigorous or formal refinement or transformation method (e.g. [Lamsweerde et al. 1995]). It is called a *derivation modelling* method, as it is not formal, but includes a set of notations together with a strategy to be followed and pragmatic heuristics.

This work assumes that it has achieved an important goal: it provides a means for constructing articulation protocols from visual and scenario-based information. These articulation protocols can then be used as a starting point for development.

7.2.3 Strengths and weaknesses of the derivation modelling method

This section describes the strengths and weaknesses of the method developed in this research, taking as its criteria the following:

- the ability to derive articulation protocols from business workflow models through scenario-based derivation modelling,

- the use of cooperation properties, such as cooperation and articulation, and

- the ability to use Use Case Maps as the visual modelling notation for these purposes

as set out in Chapters 2.3 and 3.6.5.

Derivation modelling

The first major strength of the derivation modelling method is that it combines requirements engineering modelling techniques with underlying modelling concepts of cooperation. The derivation modelling method promotes cooperation properties in an area where they are hardly used and, on the other hand, it adds clear and structured views of appropriate set of cooperation concepts. The articulation protocols resulting from the application of the derivation modelling method to a business workflow problem acts as feasible starting point for systems development trajectory where the articulation protocols can be transformed into an requirements specification and finally into a design specification.

The derivation modelling method builds on work already available for scenario-driven modelling methods. By using a visual scenario-

based technique such as Use Case Maps, the derivation modelling method produces models, which may also be used for validating the requirements.

The derivation modelling method has some weaknesses, too. Some of which can be avoided by changing parts of the method:

- For instance, the derivation modelling starts with the business workflow model. This is not a weakness in itself, but favours problems starting where actors and tasks in the business process can be described initially. If not all information is available, it may be difficult to produce an articulation protocol, as some situations in the application to the complaint management case study showed. Currently, this research proposes reverse derivation rules in capturing the services to be provided described in the articulation protocol and then use that information to construct the business workflow model.

- The use of Use Case Maps may be considered to be a weakness in the derivation modelling method in requirements engineering for business workflow systems. However, the derivation modelling method may easily be adapted to embrace other visual scenario-driven techniques. This would be a starting point for future investigation.

Importance of models used within the method

- Business workflow model: Use Case Maps are used in order to build the business workflow model. These define actors and their behaviour in a high-level way in terms of scenarios. Depending on the information available, the construction can be started by identifying candidate cooperating actors or organisational units or, if the workflow tasks are clear, the construction may begin with modelling the workflow scenario paths.

- Actor model: The actor model describes the actors' behavioural structure based on the business workflow model in terms of their tasks.

- Cooperation model: The cooperation model describes the actor relationships in terms of their dependencies. It helps to reason about the necessary services an actor provides to another actor who requires those services. The different types of dependencies describe how the cooperation dependencies between actors can be achieved.

- Articulation model: The derivation modelling method offers heuristics and derivation rules on how to build an articulation model. An articulation model is a set of articulation protocols, which defines the expectations of actors and how their cooperation dependencies as well as their tasks in the actor model can be fulfilled. These are described in terms of permitted services, guaranteed services, and rules of services for each pair of cooperating actors.

Other models that might have been used

The derivation modelling method presented in this work uses business workflow models, actor models, cooperation models, and articulation models. In addition to the four main models, it would have been possible to use further models to describe workflow properties, e.g.: a temporal scheduling model or an authorisation model. The authorisation model is briefly outlined below.

The authorisation model would describe the organisation of actors in terms of their authority status, i.e. it relates superior and subordinate actors. The actors could be placed in the authorisation hierarchy with the actor that has the highest authorisation at the top. Actors would use their authorisation relationship to allocate permissions and restrictions. The identification of the authorisation relationship says how actors communicate with each other. For example, a subordinate actor can not reject a request from a superior actor.

Since Use Case Maps, by definition, do not show interactions between components, a requirements engineering practitioner must identify if a path segment connecting two actors represents a direct or indirect relationship. In case of a direct relationship, the two actors coordinate their activities while in the case of indirect relationship, there is another actor, who facilitates the coordination.

The rules under which an actor works may change depending on the role she plays in an organisation. For example, a superior actor may disallow call waiting for a subordinate actor when the subordinate actor has the role of the help desk attendant. The identification of

roles and the actors that can fill them allows creating an authorisation hierarchy that takes into consideration all the roles actors can play.

The authorisation hierarchy may have a number of fundamental properties that are useful for modelling requirements for business workflow systems. In particular, it

- describes the resolution of conflicts. In case of a conflict, an actor may try, using the authorisation status, to resolve the conflict.

- describes the organisational mechanism for informing each other about modification, creation, or removal of any kind of resources.

Assessing Use Case Maps

While assessing business workflow models produced by using Use Case Maps in the derivation modelling method, three main questions come to mind:

- Do business workflow models based on Use Case Maps reflect the appropriate concepts needed?

- Does the derivation from business workflow model represented by Use Case Maps to other models work?

- Are heuristics for business workflow models based Use Case Maps comprehensible and repeatable by others?

Use Case Maps have a very rich set of constructs, which allow expressing many different ideas. For example, scenarios are represented as architectural entities that give a view of intertwined behaviour (sets of paths) and structure (components). Scenarios are not specified, only paths for scenarios are identified. This makes the notation more useful than stand-alone scenario notations, such as [Regnell 1999]. Causality is shown directly, avoiding the need to infer it from diagrams that express scenarios in terms of temporal sequences along timelines, such as use cases of [Jacobson 1992].

The difficulties of understanding model descriptions and specifications, in particular among stakeholders, are familiar. However, Use Case Maps are of lightweight nature and notational elements stand back from details to focus on high-level aspects. They provide a more complete scenario picture than other techniques, in the sense of being able to include more scenarios without unreasonable effort. The notation provides visual patterns for thinking and discussion about business workflow situations or systems issues.

However, Use Case Maps are by no means a complete notation for all issues that arise in business workflows or cooperative work situations. This is not the aim of Use Case Maps. Rather, the aim is to get a high-level view of structure and behaviour, because this is so difficult to achieve in practice. Use Case Maps supplement other techniques that may give more detailed views. It has been recently shown that Use Case Maps can be integrated into other software engineering methodologies and design processes, such as the

Unified Modeling Language (UML) [OMG 1999] [Amyot and Mussbacher 2001].

7.2.4 Suitability of Use Case Maps

In Chapter 3, the reasons that led to choosing Use Case Maps to model business workflow situation were discussed. The characteristics of Use Case Maps, which make it the obvious candidate, are:

- UCMs are able to produce scenario-based models,
- UCM support tools are available,
- UCMs are executable and so prototyping can be used, and
- UCMs can be integrated into other software engineering methodologies.

However, the work of Chapter 5 shows that Use Case Maps may not be ideal. They have shortcomings that will be discussed now:

- The first criticism is that Use Case Maps do not directly support cooperative work properties.
- Another criticism is that Use Case Maps, though visually displayed, do not mean a lot to stakeholders without proper documentation, which must at least include conventions and

content used. Documentation is a very important issue. Each scenario must have a name and an indication of which route is followed. Responsibilities, preconditions, postconditions can be defined textual descriptions.

In general, however, Use Case Maps are a high-level visual scenario-technique, which can stimulate thinking and discussion. It is generally accepted within requirements engineering work that this is a good thing in itself.

Even with its shortcomings, this research developed some experience with Use Case Maps, found it to be a good choice within the derivation modelling method to support the requirements engineering for business workflow systems.

7.2.5 Method applied to case studies

While developing a new analysis method, it is wise to apply it to one or more case studies. Otherwise, one cannot be sure that different types of requirements can be dealt with appropriately within the new approach.

For this work, it was important to apply the method to one minor case study (i.e. the meeting scheduler problem) and one major case study (i.e. complaint management in a bank). With the application to the meeting scheduler problem, the method is validated conceptually. The complaint management case study was chosen for this research

to validate and improve the method, since it represents a real-world problem and not one, made up in a research laboratory. By applying the method to one minor and one major case study, a variety of aspects were identified, which improved the method substantially. The application to one or more other case studies could have identified further characteristics for further enhancement of the method. However, it is believed that the application to other case studies would not have resulted in further fundamental insights with regard to the objectives set out for this work.

7.3 Summary

This chapter presents the principal ideas behind this work. Section 7.1 presents the evolution of the method, justifying the current version. The derivation modelling method developed in this thesis is a result of successive refinements and the effort to encapsulate cooperative work aspects into this method:

- Concepts were provided that describe dependencies between actors and the various mismatches that might exist.
- Concepts for intention and behaviour were incorporated.
- Graphical concepts for modelling commitment and expectations between cooperating actors were used.
- Articulation was taken into account as a major concept for the method.

The most significant problems to be solved in order to develop the method to its current state were to:

- to define an appropriate set of model, which includes the necessary aspects – workflow, cooperation, collaboration, and articulation,
- to define articulation protocols as permitted or guaranteed services and rules for these services,
- to define derivation modelling rules and appropriate heuristics and guidelines,
- to define the use of scenarios in this method, and
- to define cooperation dependencies for cooperating actors.

Section 7.2 begins by discussing properties that a requirements engineering method for modelling requirements for business workflow systems should display:

- the ability to derive articulation protocols from business workflow models through scenario-based derivation modelling,
- the ability to use articulation protocols as a starting point for development, and
- the ability to use Use Case Maps as the visual modelling notation for these purposes.

Then, it evaluates the work, by discussing its weaknesses and strengths of derivation modelling, the importance of the models used within the method, and the choice in using Use Case Maps for visual representation for business workflow models.

8 Summary and Conclusions

This chapter summarises the work of this thesis, revisits the solution as proposed in the first chapter, and suggests possibilities for future work.

8.1 Thesis summary

The work in this thesis presents a novel contribution in the area of requirements engineering for business workflow systems by developing and evaluating a scenario-based derivation modelling method. It is motivated by the fact that workflow implementations require a deep understanding of business and human cooperation. Previous approaches have addressed this need for understanding, but to a large extent in a descriptive and analytical manner. Various attempts to use such approaches in software development have had mixed results.

This thesis presents a modelling method, which has been developed to support the requirements engineering process with properties of cooperation and integrating these through the use of a derivation modelling approach. The provision of pragmatic heuristics and guidelines supports real-world requirements engineering practitioners and thus ensure a high probability of success for the business workflow system to be developed.

This method provides clear and structured views of cooperation properties such as collaboration and articulation, and allows the derivation of articulation protocols from business workflow models, allowing to define how the expectations of the cooperation between actors are to be fulfilled by a system. This provides a statement of requirements for business workflow systems that reflects the richness of these and also acts as a feasible starting point for development.

As indicated in Chapter 1, three major results can be stated from this thesis: the modelling properties for cooperation, the derivation modelling method, and pragmatic heuristics and guidelines.

8.1.1 Properties of cooperative work

The various properties of cooperative work being used in the derivation modelling method are intended to capture the highly complex situations of workflow scenarios in business and human cooperation. These properties are mainly collaboration and

articulation. Collaboration is the process where actors cooperate while they produce some product or service. This process is often a negotiation process. The process results in one unified result of all the contributions made by the individual actors.

Articulation is the work about cooperative work. An Articulation model describes a set of articulation protocols, which define the expectations of cooperating actors and how their objectives can be successfully fulfilled in terms of permitted and guaranteed services.

8.1.2 Derivation modelling method

The derivation modelling method presented in Chapters 4 and 5 provides a means for constructing articulation protocols from visual and scenario-based business workflow models.

It builds on already available work for scenario-driven modelling methods. In this work, Use Case Maps are used to produce business workflow models and architectural solutions, which can also be used for validation of requirements.

8.1.3 Pragmatic heuristics and guidelines

Chapter 6 is devoted to the provision of pragmatic heuristics and guidelines for requirements engineering practitioners. This method

provides heuristics for the construction for each of the models and the derivation of one model from one another.

The lessons learnt are emphasised in applying the meeting scheduler problem, which serves as the origin for the ideas to formulate this method. The heuristics defined in Chapter 6 in this thesis were derived from experiences and observations during the application of the derivation modelling method to the complaint management case studies.

8.2 Proposed solution revisited

The proposal made in Chapter 1 states the following:

"A derivation modelling method will:

- *provide clear and structured views of cooperation properties,*
- *allow the derivation of articulation protocols from business workflow models in a scenario-driven manner,*

and so provide requirements engineering to define how the expectations of the cooperative situation are to be fulfilled by the workflow systems to be developed."

The needs of a derivation modelling method are defined in Chapter 3. The method has the following characteristics:

- It produces a set of articulation protocols, which includes important properties that reflect cooperative work settings.

- It proposes methodological steps to be followed when constructing the intermediate models and final models (the business workflow model is based on Use Case Maps; the actor model and the cooperation model can be produced from information in the business workflow model; the articulation model can be created by following the heuristics).

- It provides a development process, by offering a set of derivation rules, heuristics, and mappings from the business workflow model to articulation protocols.

Chapter 5 shows that the separation of tasks from responsibilities helps to make the business workflow model using UCMs more comprehensible and more precise for deriving the actor and cooperation model. The resulting business workflow model affords the ability to construct a more comprehensible model, while facilitating the construction of more complete and realistic derived models.

Moreover, the inclusion of a collaboration dependency in the cooperation model ensures that collaborative situations are explicitly considered in articulation protocols by use of a generic negotiation mechanism. The articulation protocol is to define expectations of how actors can fulfil cooperation dependencies as well as the tasks they have defined by the actor model. This distinction between the

cooperative interaction and the articulation protocol offers a clear separation of concerns for the interaction that must occur and by the services it is achieved. These examples are shown in Chapter 5.

However, when applied to the case study in Chapter 5, some limitations of the derivation modelling method are noticeable, as outlined in Chapter 7.2.3. These limitations have two aspects:

- The derivation modelling starts with the business workflow model. This is not a weakness in itself, but favours problems starting where actors and tasks in the business process can be described initially. If not all information is available, it may be difficult to produce an articulation protocol, as some situations in the application to the complaint management case study showed.

- The use of Use Case Maps may be considered to be a weakness in the derivation modelling method in requirements engineering for business workflow systems. The method should be easily adapted to embrace other visual scenario-driven techniques.

It can be concluded that the development and evaluation of the proposed solution results in contributions to the area of requirements engineering for systems that support cooperative work, in particular business workflow systems. The limitations lead to considerations for further work, discussed briefly in the next section.

8.3 Future work

The limitations revealed by the evaluation in Chapter 7 point out that the development of the derivation modelling method is not complete. There are three main areas in which future work could usefully be carried out: improvement of the method itself, useful tools to support the method, and broader applications.

8.3.1 Improvements of the method

One area of further investigation is concerned with reverse derivation modelling. The derivation modelling method needs more flexibility in the sense that the manipulation in one model has an effect on the others. The effects are defined in the derivation rules. For example, a change of a service in the articulation protocols must result in the appropriate change in the business workflow model.

A related area of work would be to Use Case Maps so that other scenario-based modelling language, such as UML, can be used. This would involve refining the derivation rules.

8.3.2 Tools support

The development of a software tool can support the application of the derivation modelling method. This can be seen twofold: In a first step, the tool can provide support that helps the requirements

engineering practitioner to derive the articulation protocols based on heuristics. The systematic approach that is provided by the derivation modelling method supports the practitioners, with tool assistance, to manipulate the business workflow model and derive the next model until articulation protocols are derived.

In a second step, the tool would support traceability between the various models. The practitioner would be able to trace the chain of derivation forward and backward from any part of any model. Also, a change in any part of any model would cause all effected attributes to be highlighted. This aids the practitioner in identifying places where changes should be made and ensuring the models remain consistent.

8.3.3 Further applications

During the time of this research, the method was applied and improved by applying it to case studies. However, another application could be explored to a type of problem, which is of increasing importance today – the area of e-commerce systems.

In the development of e-commerce systems, a justification of the business idea needs to be established to build up confidence among stakeholders in the feasibility of the idea. The strong relationship between the business workflow model and articulation protocols could be measured in terms of cost and profit drivers, which leads to an extension of the method. For example, each task performed by

an actor has a certain value. Each articulation protocol is a service for another actor to be provided, whereby each service has a certain value, either for themselves or for others, too. An estimation of the cost and profit per scenario and/or per actor can be taken into account, which can be obtained from the articulation model. This extension of the method would help stakeholders to understand the direct impact of their business workflow model in terms of quantifications of profits and costs during the requirements engineering phase.

8.4 Concluding remarks

In summary, this thesis demonstrates that it is possible to derive articulation protocols from business workflow scenarios by means of a derivation modelling method in a practical and effective way and so to accommodates for human and business cooperation properties. It is believed that it is possible to apply the method to create initial requirements models, which can then be used as a feasible starting point of a business workflow system development strategy.

Appendix A - Glossary

This appendix gathers a glossary of the main terms that are used in this thesis in a non-standard way, or used to refer to specific features of the method presented in this work.

Actor model	The actor model describes the behavioural structure of the actors discovered in the business workflow model. The actor model is derived from the high-level workflow model and is described in terms of their goals and tasks.
Articulation protocol	An articulation protocol defines expectations of how actors can fulfil cooperation dependencies as well as the tasks they have defined by the actor model. An articulation protocols is described in terms of permitted and guaranteed services and the rules for these services.
Business workflow model	A business workflow model identifies actors and their behaviour. It gives a high-level view of the actors and workflows, and provides a starting point for deriving the details of the

other models.

Collaboration Collaboration requires actors to work together to achieve a common goal, under the condition that a contribution is needed by each participating actor.

Cooperation model A cooperation model describes actor relationships in terms of cooperation dependencies.

Derivation modelling Derivation modelling proposes methodological steps to be followed when constructing the intermediate models and final models. It provides a development process, by offering a set of derivation rules, heuristics, and mappings from the business workflow model to articulation protocols.

Heuristic Heuristics are sets of rules, which guide the requirements engineering practitioner in the identification and analysis of requirements and for construction of the different models. They can be considered as a collection of hints and rules-of-thumb and may be applied wherever they make sense.

Method A method is a generic guide to help performing some activity. This work presents

a method, which applies for doing requirements engineering for business workflow systems. Typically, a method consists of the following components:

- a set of modelling concepts for capturing semantic knowledge
- a set of views and notations for presenting underlying modelling information to people that allow understanding them
- a development process for constructing models, which may be described at various levels of details, from overall management down to specific steps of how to build low-level models
- a collection heuristics and guidelines, which are not necessarily organised into steps to be followed, but may be applied wherever they seem useful

Model

A model is a representation of the world in which the problem is located, described at a

certain level of abstraction. It is built out of a collection of modelling concepts, which seem most useful for describing requirements of the application domain.

Scenario

A scenario is a description of the world in a particular context, including the structure and behaviour of actors and sufficient context information. It is intended as a means of communication among stakeholders, and to constrain requirements engineering from one or more perspectives.

Appendix B – Use Case Maps Overview

This appendix presents the Use Case Maps as used in this thesis.

UCM [Buhr and Casselman 1995] is a high-level scenario modelling technique defined for real-time object-oriented system design. It is based on a simple and expressive visual notation that allows describing scenarios at an abstract level in terms of causal sequences of responsibilities over a set of components.

The primary objective of the UCM technique is to capture, model, and analyse system requirements and behaviour at an abstract level. The technique supports individual scenario descriptions, scenario interactions, responsibility allocations and, inter-component communication.

UCM also provides important features, such as:

- Superimposition of scenarios on system structure: This enables to visualise scenarios in the context of system structure for architectural reasoning. It also provides a mechanism by which responsibilities can be allocated to system components.

- Combining sets of scenarios in a single diagram: This enables to express scenario clusters and scenario interactions in a graphical

manner. It also provides a mechanism that can be used to analyse the overall system behaviour that emerges from scenario combinations.

The next sections describe the UCM terminology and notation used as a basis in this thesis. Additional notations required in the different examples and case studies will be described as they are used.

A *use case path* represents a path along which scenarios flow in a system. They express the sequences of responsibilities that need to be performed by system components in order to achieve the overall objective of the system in response to a given triggering event.

This section describes the basic notation used to describe paths, and then it is described.

In Figure 13, the basic elements that compose a use case *path* are illustrated.

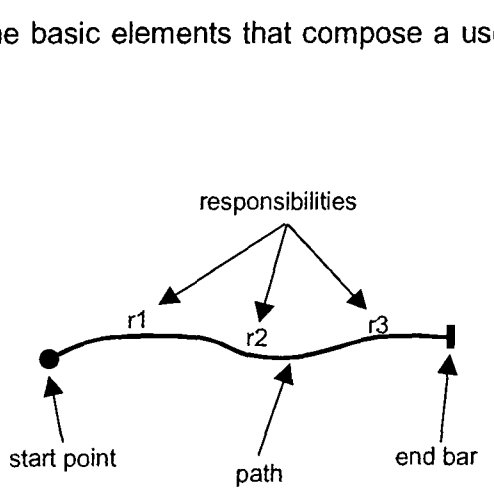


Figure 13: A simple UCM path

The following sections describe the elements of a use case path - start point, responsibilities, scenario, end bar, and path.

The performance of a use case path begins at a *start point*. A start point is illustrated in UCM by means of a filled circle placed at the origin of a scenario (see Figure 13).

A start point is defined by means of possible triggering events (stimulus) and maybe a precondition. If a precondition is specified, this precondition must be true to perform the path.

A use case path describes a sequence of *responsibilities* that need to be performed by components in response to a given stimulus. At the UCM modelling level, these responsibilities are high-level ones. Thus, responsibilities are informal elements of a model that are usually more precisely defined in later stages of the development process.

Responsibilities are visually illustrated in UCMs by means of named, short, prose descriptions (r1, r2, and r3 in Figure 13) of some actions along paths. Whether a responsibility point is visible or not along a path, the existence of at least one is always implied. To avoid the creation of cumbersome UCMs, the responsibility points that are placed along the path are usually short identifiers, i.e. one, two or, three characters (letters and digits).

Two responsibilities along a path have a *cause-effect relationship.* The original cause is the stimulus. The next effect is that the first responsibility along the path is performed. This in turn is a cause

relationship to the next responsibility point along the path after that, etc., as the causes accumulate to result in each next effect.

The path ends where the ultimate effect happens. A path is progressive in the sense that each responsibility point along a path advances the path towards an end. The cause-effect relationship is a property of each path and the preconditions that cause it. If there is a cause-effect relationship between two responsibility points along one path, this does not mean that they have the same relationship along another path.

This work considers responsibilities as prose descriptions. Responsibilities may also be expressed in some formal language that treats them like states of the underlying system and the transformation of preconditions into postconditions by series of responsibilities.

A *path segment* expresses an ordered sequence of path elements (such as a responsibility or a waiting place) that need to be performed by components. It is visually illustrated by means of a "wiggly" line joining together the sequence of path elements. Path segments show the operation of the components, but do not model the way in which responsibilities change the system state, cause information flow, etc.

A *use case* (e.g. in terms of Ivar Jacobson's approach [Jacobson 1992]) is a prose description of a path segment or of a set of them of a user's interactions with a system seen as a black box.

The performance of a path terminates at an *end bar*. A thick rectilinear line placed at the end of a path visually illustrates an end bar in an UCM. An end bar is defined by means of some resulting event or postcondition (effect).

A *path* may have any shape as long as it is continuous. It is composed of one of more coupled path segments. Although a path may be able even to cross itself, this can create visual ambiguity related to other aspects of the notation.

A basic path as a complete unit of a map is a path with a start point (in general represented by a waiting place) and an end presented by a bar. In addition, the direction of a path may be indicated in complicated or fragmented maps by an arrow to show directions.

UCMs consist of paths that traverse one or more components and therefore are a means of explicitly linking views of behavioural patterns of systems. Maps with no visible components are called *unbound maps* (Figure 14) and maps with visible components along their paths are called *bound maps* (Figure 15).

Unbound maps provide a visual notation for use cases. They are useful for illustrating transitions at the highest level of abstraction.

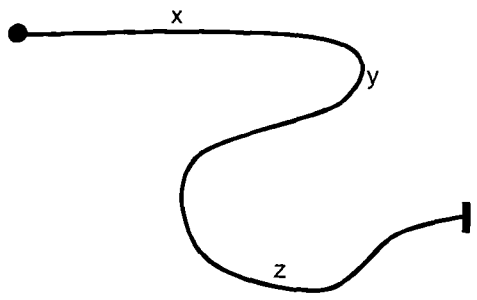


Figure 14: Unbound map

Bound maps show how a system's components contribute jointly to achieve properties of the environment. Components are visually illustrated using labelled rectangles. At this level, the system structure is only defined as a set of components. Interaction among components is not yet defined.

In bound use case maps, responsibilities are allocated to components. For example in Figure 15, responsibility x is allocated to component B, responsibility y is allocated to component C, and responsibility z is allocated to component D.

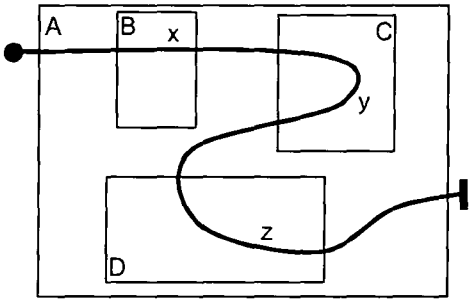


Figure 15: Bound map

The boxes used so far for illustrating components in this section in use case maps are useful as representations of components of uncommitted types. In order to be able to define components and make judgement about the architecture of a system, the most important ones are introduced; they are divided into static and dynamic components.

The following static component types are described: teams, processes and, objects:

- *Teams*: Teams are abstractions for components at the level of use case maps. They are mainly introduced into maps to hide details without committing to whether they will actually exist as components with interfaces or will actually have members. In general, a team is an operational grouping of components that may include members of any or all of objects, processes, other teams, etc.

- *Processes*: Processes are autonomous components that may operate concurrently with other processes. A process has no other concurrent elements inside, the only concurrent elements are the processes themselves.

- *Objects*: Objects perform their own responsibilities but do not have ultimate control of when they perform them. The control comes from processes, although it may also come indirectly through other non-process components, such as teams or other objects. In use case maps, objects are not further decomposable into teams of finer objects.

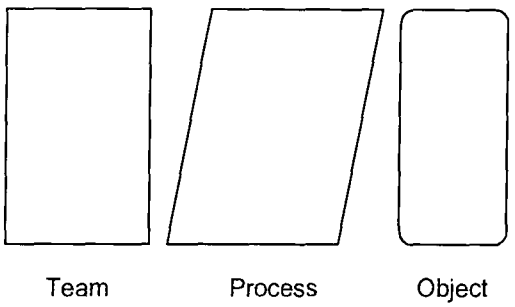


Figure 16: Static components

The following dynamic component types are described: slots and pools (Figure 17):

- *Slots*: Slots are organisational components that may be temporarily occupied by different dynamic components (one at a time) or are empty. Slots are *fixed* components in maps in the sense that they are assumed to have fixed positions and fixed responsibilities along paths that they traverse. Occupants of slots are assumed to be able to fulfil the required slot responsibilities.

- *Pools*: Pools are placeholders for *dynamic* components with the aptitude to move into slots. A dynamic component is one that may be created and destroyed at any time during the lifecycle of the map that has a slot for it, and may move in or out of this slot at any time. Paths may not drawn across pools to indicate the performance of responsibilities along the path.

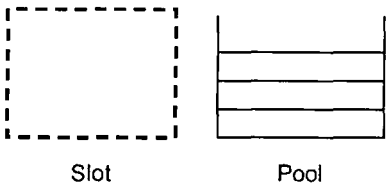


Figure 17: Dynamic components

Slots and pools are sources and destinations of transitions into and out of paths. The possibility of dynamic components being created or destroyed along paths and of them moving into, along, and out of paths can be included in use case maps. Buhr and Casselman [Buhr and Casselman 1995] suggest to use suitably annotated small arrows with either their heads or their tails touching paths (Figure 18):

- *Move*: Used for unaliased moves from a path to a slot, or vice versa.

- *Move-stay*: Used for aliased moves.

- *Create*: The component moved is created before the move. Initialisation is assumed to be part of the create responsibility.

- *Destroy*: The component moved is destroyed after the move.

- *Copy*: This is similar to move-stay, except that, instead of moving the same component, a copy of it moves.

Their notation offers both *unaliased* and *aliased* moves. An unaliased move is the default. The aliased move ends up with the same component in more than one slot. An aliased move may be compared with a human organisation, where a person can play different roles at the same time. Aliasing is different from copying, which results into different but identical components in more than one slot.

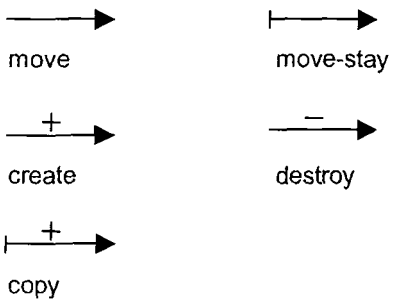


Figure 18: Movement notation of UCM for dynamic components

The example below (Figure 19) illustrates the creation of a single component along the path, ends up aliased in slots S1 and S2 and another component that was in the pool ends up in slot S3.

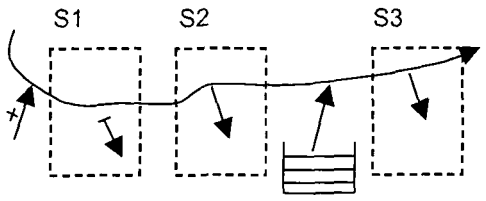


Figure 19: Creation of a single component along a path

In the previous sections, purely sequential paths have been used. UCM offers also more complex cases that involve concurrent or alternative path coupling constructs.

The UCM technique uses path segment coupling with the following constructs: AND-fork, AND-join, OR-fork, and OR-join. These coupling constructs are illustrated in Figure 20. In this figure, each path segment is labelled with a different responsibility point. The performance of the four path diagrams given in this figure goes from left to right.

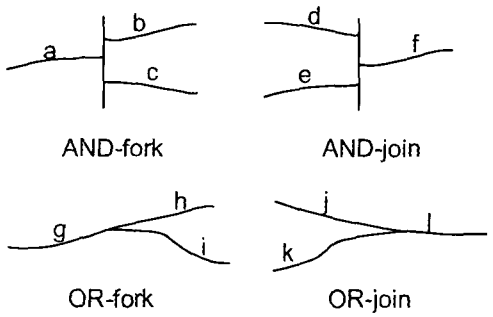


Figure 20: Path segment coupling

In the following sections, each of the scenario coupling constructs is described in more detail.

An AND-fork is used to illustrate a point along a path where the performance of a single scenario splits into two or more concurrent paths that may proceed independently and, if concurrency is allowed, concurrently.

Once the performance of a path is complete, then the concurrent performance of paths b and c may start.

An AND-join is used to illustrate a point along a path where several concurrent scenarios synchronise together and result in the performance of one path.

Once the performance of scenario d and e is finished, then the performance of path segment f may start.

An OR-fork is used to show a point along a path where alternative branches may be followed. Each branch is associated with a distinct path segments.

Once the performance of scenario g is finished, then the performance of scenario h or i will be triggered.

An OR-join is used to illustrate a point along a path where two or more incoming scenarios merge into a single one without requiring any synchronisation or interaction between the incoming path segments.

The performance of either scenario j or k will result in the performance of path l. Thus, the OR-join diagram illustrates two possible paths: one formed by path segments j-l, and one formed by path segments k-l.

This set of path segment coupling constructs which have been described in the above sections can be combined together to describe more complex paths scenarios. Some examples of the type of path constructions that can be described by combining these constructs are illustrated in Figure 21.

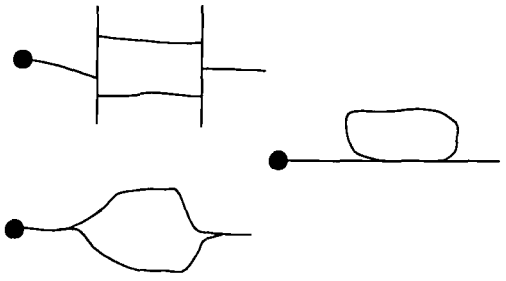


Figure 21: Combination of path segment coupling

Two other types of the UCM notations of Buhr are used in this thesis: waiting places and static and dynamic stubs.

Waiting places are used to indicate a point along a path where the progression of the path is blocked until a predefined event occurs. Two different types of waiting places can be identified: a regular waiting place, and a timer. The according notation is illustrated in Figure 22 and is describes below.

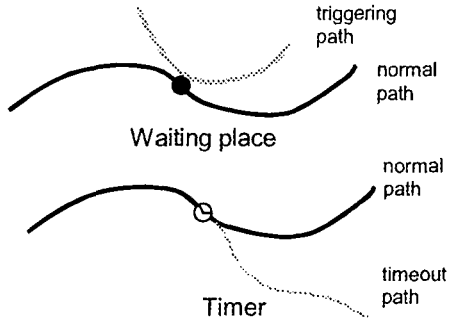


Figure 22: Waiting places

A waiting place is identifies a point along the normal path at which the progression of a path is blocked until an event occurs, e.g. by a triggering path. Visually, waiting places are illustrated using filled circles place along a path. Waiting places are used to illustrate

points along paths where interactions with other paths or with the environment of a system occur. They may be associated with both synchronous and asynchronous interactions. The starting point constitutes a special use of a waiting place.

A timer is a special type of waiting place that will only wait for a certain period before the scenario continues. If the timer runs out, before a trigger occurs it proceeds at the timeout path. If the trigger occurs before the time out, the scenario proceeds at the normal paths. A clock-like icon placed along a path visually represents timers.

The UCM modelling technique provides a mechanism for path abstraction, called *stub*. A stub illustrates part of a path that is abstracted in the context of a use case map in which it is used in order to defer details. In an use case map, the expansion of a stub is either described in separate maps (called *plug-ins*), or remains to be defined at a later stage when details will be added to the map. Stubbing constitutes an important mechanism for iterative development. It also reduces confusion of models by hiding details that are less important in the context of a given map.

The stub in Figure 23 is static (a static stub would be indicated by a dashed outline), since no dynamic selection of different plug-ins is implied. The plug-in in the enclosed circle has been collapsed into a stub.

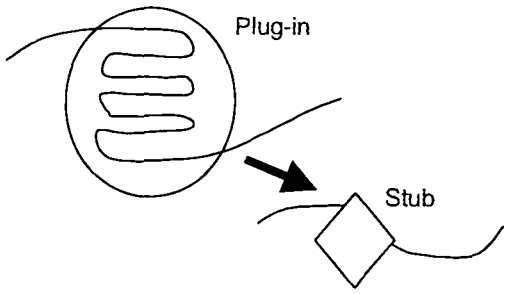


Figure 23: Static stub

Stubs were originally viewed in [Buhr and Casselman 1995] as static decomposition technique for paths. The concept of dynamic stubs and plug-ins is a new one relative to earlier UCM work [Buhr 1998]. The stub in Figure 24 is dynamic in the sense that the available plug-ins can be selected dynamically when a scenario arrives at the stub.

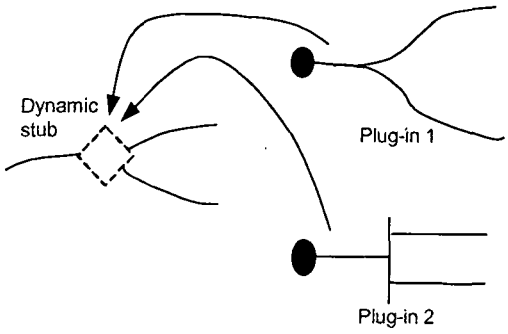


Figure 24: Dynamic stub with multiple plug-ins

If a scenario arrives at the dynamic stub, one of the plug-ins is selected based on a predefined precondition.

In this thesis, a set of paths segments of a use case map that can be triggered from a single starting point is called *scenario ensemble*. A scenario ensemble is composed of a starting point, a set of paths, a

set of coupling constructs, and a set of end bars (each indicating a distinct path termination).

An abstract example of a scenario ensemble is given in Figure 25.

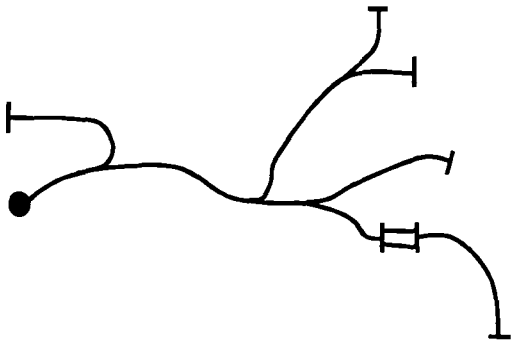


Figure 25: A UCM scenario ensemble

A scenario ensemble constitutes a cohesive logical entity, since it can be started from a single triggering event and it is a set of paths.

An important aspect of the Use Case Map modelling technique is that it allows describing asynchronous and synchronous path interactions.

In this thesis, two asynchronous types of interaction are used: *trigger after completing path performance* and *trigger in passing.*

Trigger after completing path performance is used to illustrate cases where the completion of the performance of a path triggers another path that is waiting on a waiting place. The waiting can be either a start point or a waiting place along a path. Both cases are shown in Figure 26.

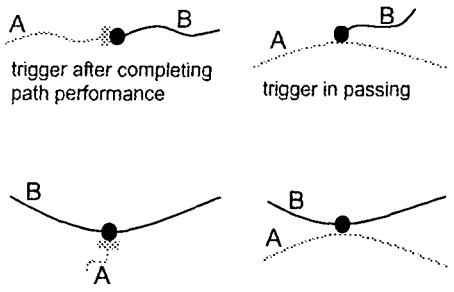


Figure 26: Asynchronous path interactions

Trigger in passing is used to illustrate cases where a waiting place is triggered by another path in an asynchronous manner.

Three types of synchronous interaction are used in this thesis: AND-join, rendezvous, synchronisation and, abort (Figure 27).

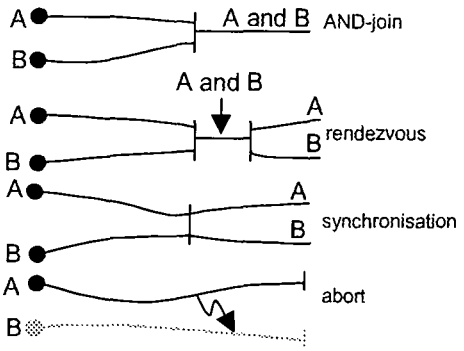


Figure 27: Synchronous path interactions

The *AND-join* is used to illustrate cases where the synchronisation of two or more paths results in the performance of one.

Rendezvous is used to illustrate cases where two or more paths synchronise together to perform a certain scenario (sequence of responsibilities) before returning to the performance of their own respective path.

Synchronisation is used to illustrate cases where two or more paths synchronise together and then return to the performance of their own respective path.

Abort notation is used to illustrate cases where the performance of a path interrupts the performance of another.

An important feature of use case maps is that several scenario ensembles can be coupled into a more complex diagram, called *composite use case map*. This allows expressing scenario interactions and concurrency.

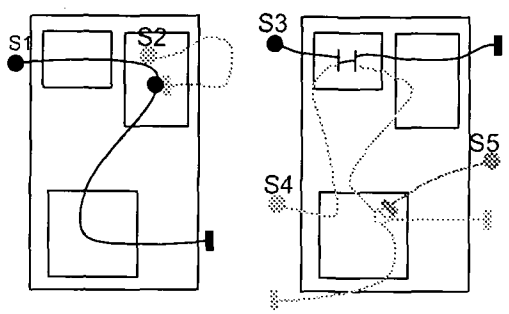


Figure 28: Composite use case maps

Figure 28 illustrates two abstract examples of composite use case maps. In the left example, scenario S1 triggers scenario S2 in passing, and then waits for the completion of scenario S2 before continuing its performance. In the right example, a more complex inter-scenario relationship composite map between scenarios S3, S4, and S5 is given. This example shows the use of a synchronous interaction and a timer.

A UCM model is composed of a set of UCM maps and a set of UCM scenario ensembles. A UCM map is either a simple or a composite map, which describes relationships among scenarios. So, a UCM map is composed of paths, paths of interactions, components and may be some responsibility allocations that links responsibilities to components.

Scenario ensembles constitute the building blocks of UCM maps. They are the basic elements from which composite maps are built. It should be noted that a scenario ensemble can be involved in several UCM maps to illustrate different scenario relationships. Each path in a UCM model is contained in a scenario ensemble.

In Figure 29, the definition of a composite map from two scenario ensembles is illustrated.

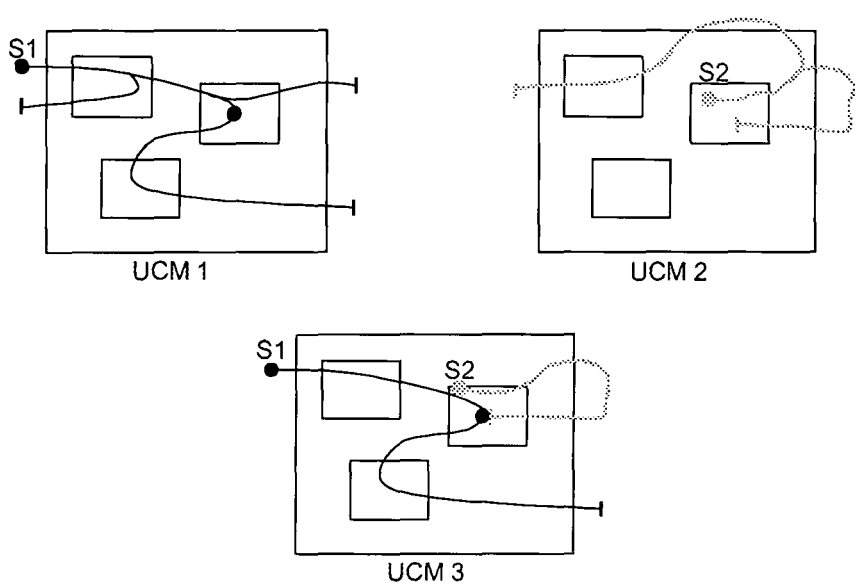


Figure 29: Scenario ensemble and composite maps

UCM 1 and UCM 2 are two simple maps that each contains a scenario ensemble. UCM 3 is a composite map that defines a relationship between scenario S1 and scenario S2.

References

Aalst, Wv, Basten, T, Verbeek, H, Verkoulen, P and Voorhoeve, M 1998. Adaptive Workflow - On the interplay between flexibility and support. Paper Presented at *First International Conference on Enterprise Information Systems*, Setubal, Portugal.

Alonso, G, Agrawal, D, Abbadi, AE and Mohan, C 1997. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert* 12(5).

Alonso, G, Fiedler, U, Hagen, C, Lazcano, A, Schuldt, H and Weiler, N 1999. Wise: Business to Business E-Commerce. Paper Presented at *IEEE 9th International Workshop on Research Issues on Data Engineering. INFORMATION TECHNOLOGY FOR VIRTUAL ENTERPRISES (RIDE-VE'99)*, Sydney, Australia.

Amyot, D and Mussbacher, G 2001. Bridging the Requirements/Design Gap in Dynamic Systems with Use Case Maps (UCMs). Paper Presented at *23rd International Conference on Software Engineering (ICSE'01)*, Toronto, Canada.

Antón, A 1996. Goal-Based Requirements Analysis. Paper Presented at *Second IEEE International Conference on Requirements Engineering (ICRE'96)*, Colorado Springs, Colorado.

Bannon, L and Schmidt, K 1991. CSCW: Four Characters in Search of a Context. In *Studies in Computer-Supported Cooperative Work: Theory, Practice and Design* (Bowers, J and Benford, S), 3-16.

Bardram, J 1997. Plan as Situated Action: An Activity Theory Approach to Workflow Systems. Paper Presented at *Proceedings of the 5th European Conference on Computer Supported Cooperative Work (ECSCW '97)*, Lancaster, UK.

Bardram, J 1998. *Collaboration, Coordination and Computer Support: An Activity Theoretical Approach to the Design of Computer Supported Cooperative Work.* PhD thesis, Department of Computer Science, University of Aarhus.

Barros, A, Hofstede, At and Proper, H 1997. Essential Principles for Workflow Modelling Effecitiveness. Paper Presented at *Third Pacific Asia Conference on Information Systems (PACIS'97)*, Brisbane, Australia.

Boehm, B 1988. A Spiral Model of Software Developoment and Enhancement. *IEEE Computer* 21(2): 61-72.

Bogia, D and Kaplan, S 1995. Flexibility and Control for Dynamic Workflows in the wOrlds Environment. Paper Presented at *Conference on Organisational Computing Systems*, Milpitas, CA.

Bowers, J, Button, G and Sharrock, W 1995. Workflow from Within and Without: Technology and Cooperative Work on the Print Industry Shopfloor. Paper Presented at *Fourth European Conference on*

Computer Supported Cooperative Work (CSCW'95), Stockholm, Sweden.

Buhr, R 1998. Use Case Maps as Architectural Entities for Complex Systems. *IEEE Transactions on Software Engineering, Special Issue on Scenario Management* 24(12): 1131-1155.

Buhr, R and Casselman, R 1995. Use Case Maps for Object-Oriented Systems. USA: Prentice-Hall.

Carroll, JM 1995. Introduction: The Scenario Perspective on System Development. In *Scenario-Based Design: Envisioning Work and Technology in System Development* (Carroll, JM), 1-17.

Cockburn, A 1997. Using goal-based use cases. *Journal of Object-Oriented Programming (JOOP)* 10(7): 56-62.

Curtis, B, Krasner, H and Iscoe, N 1988. A Field Study of the Software Design Process for Large Systems. *Communications of ACM* 31(11): 1268-1287.

Dardenne, A, Lamsweerde, Av and Fickas, S 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20(1-2): 3-50.

Dourish, P, Holmes, J, MacLean, A, Marqvardson, P and Zbyslaw, A 1996. Freeflow: Mediating Between Representation and Action in Workflow Systems. Paper Presented at *Conference on Computer Supported Cooperative Work (CSCW'96)*, Cambridge, MA.

Ellis, C, Keddara, K and Rozenberg, G 1995. Dynamic change within workflow systems. Paper Presented at *Organizational Computing Systems*, Milipitas, CA.

Feather, M, Fickas, S, Finkelstein, A and Lamsweerde, Av 1997. Requirements and Specification Exemplars. *Automated Software Engineering* 4(4): 419-438.

Fitzpatrick, G 1998. *The Locales Framework: Understanding and Designing for Cooperative Work*. PhD Thesis, Department of Computer Science and Electrical Engineering, The University of Queensland.

Fitzpatrick, G, Kaplan, S and Mansfield, T 1996. Physical spaces, virtual places and social worlds: A study of work in the virtual. Paper Presented at *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '96)*, Boston, Mass.

Floyd, C 1987. Outline of a Paradigm Change in Software Engineering. In *Computers and democracy - a Scandinavian challenge* (Bjerknes, G et al.), 191-212.

Georgakopoulos, D, Hornick, M and Sheth, A 1995. An Overview to Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases* 3 119-153.

Gerson, E and Star, SL 1986. Analyzing Due Process in the Workplace. *ACM Transactions on Office Information Systems* 4(3): 257-270.

Goguen, J 1992. The dry and the wet. In *Information Systems Concepts* (Falkenberg, E et al.), 1-17.

Goguen, J 1994. Requirements Engineering as the Reconciliation of Technical and Social Issues. In *Requirements Engineering: Social and Technical Issues* (Goguen, J and Jirotka, M), 165-199.

Goguen, J 1996. Formality and informality in requirements engineering. Paper Presented at *International Conference on Requirements Engineering (ICRE'96)*, Colorado Springs, Colorado.

Grudin, J 1988. Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. Paper Presented at *Conference on Computer-Supported Cooperative Work (CSCW'88)*, Portland, Oregon.

Grudin, J and Palen, L 1995. Why Groupware Succeeds: Discretion of Mandate? Paper Presented at *Fourth European Conference on Computer-Supported Cooperative Work (ECSCW '95)*, Stockholm, Sweden.

Haumer, P, Heymans, P, Jarke, M and Pohl, K 1999. Bridging the Gap Between Past and Future in RE: A Scenario-Based Approach. Paper Presented at *Fourth IEEE International Symposium on Requirements Eng. (RE'99)*, University of Limerick, Ireland.

Haumer, P, Pohl, K and Weidenhaupt, K 1998. Requirements Elicitation and Validation with Real World Scenes. *IEEE Transactions on Software Engineering* 24(12): 1036-1054.

Heath, C and Luff, P 1992. Collaborative Activity and Technological Design: Task Coordination in London Underground Control Rooms. *Computer Supported Cooperative Work* 1(1-2): 69-94.

Hollingsworth, D 1995. Workflow Management Coalition: The Workflow Reference Model. *The Workflow Management Coalition Specification* (TC00-1003).

Hughes, J, Randall, D and Shapiro, D 1992. Faltering from Ethnography to Design. Paper Presented at *Proceedings of the Conference of Computer Supported Cooperative Work (CSCW '92)*, Toronto, Canada.

Jackson, M 1995. Software Requirements and Specifications.Addison-Wesley.

Jackson, M 1997. The meaning of requirements. *Annuals of Software Engineering* 3 5-21.

Jackson, M and Zave, P 1995. Deriving Specifications from Requirements: an Example. Paper Presented at *International Conference on Software Engineering (ICSE-17)*, Seattle, USA.

Jacobson, I 1992. Subject-Oriented Software Engineering - A Use Case Driven Approach. New York: Addison-Wesley.

Jacobson, I 1995. The Use-Case Construct in Object-Oriented Software Engineering. In *Scenario-based Design - Envisioning Work and Technology in System Development* (Carroll, J), 309-336.

Jarke, M, Bui, XT and Carroll, J 1999. Scenario Management: An Interdisciplinary Approach. *Requirements Engineering Journal* 3(3-4): 154-173.

Jordan, B 1996. Ethnographic Workplace Studies and CSCW. In *The Design of Computer Supported Cooperative Work and Groupware Systems* (Shapiro, D et al.).

Karat, J 1995. Scenario Use in the Design of a Speech Recognition System. In *Scenario-based Design - Envisioning Work and Technology in System Development* (Carroll, J), 109-134.

Kling, R 1992. Cooperation, coordination and Control in Computer Supported Cooperative Work. *Communications of the ACM* 34(12): 83-88.

Koksal, P, Cingil, I and Dogac, A 1999. A Component-based Workflow System with Dynamic Modifications. Paper Presented at *Next Generation Information Technologies and Systems (NGITS'99)*, Israel.

Kuutti, K 1995. Work Processes: Scenarios as a Preliminary Vocabulary. In *Scenario-based Design - Envisioning Work and Technology in System Development* (Carroll, J), 19-36.

Kyng, M 1995. Creating contexts for Design. In *Scenario Based Design: Envisioning Work and Technology in System Development* (Carroll, J), 85-108.

Lamsweerde, Av, Darimont, R and Massonet, P 1992. *The Meeting Scheduler System - Problem Statement.* Université Catholique de Louvain, Département d'Ingénierie Informatique, Louvain-la-Neuve (Belgium).

Lamsweerde, Av, Darimont, R and Massonet, P 1995. Goal -Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. Paper Presented at *International Symposium on Requirements Engineering (ISRE '95)*, York, UK.

Lubars, M, Potts, C and Richer, C 1993. A review of the state of the practice in requirements modeling. Paper Presented at *Symposium on Requirements Engineering*, San Diego, CA.

Luff, P, Heath, C and Greatbach, D 1992. Tasks-in-interaction: paper and screen based documentation in collaborative activity. Paper Presented at *Conference of Computer Supported Cooperative Work (CSCW '92)*, Toronto, Canada.

Macaulay, L 1993. Requirements Capture as a Cooperative Activity. Paper Presented at *IEEE International Symposium on Requirements Engineering*, San Diego, California.

Macaulay, LA 1996. Requirements Engineering. London, UK: Springer Verlag.

Mack, R 1995. Discussion: Scenarios as Engines of Design. In *Scenario-based Design - Envisioning Work and Technology in System Development* (Carroll, J), 361-386.

Malone, T and Crowston, K 1994. The Interdisciplinary Study of Coordination. *ACM Computing Surveys* 26(1): 87-119.

McCready, S 1992. There is more than one kind of Work-flow Software. *Computerworld*.

McGraw, K and Harbison, K 1997. User Centered Requirements, The Scenario-Based Engineering Process. Mahwah, New Jersey, USA: Lawrence Erlbaum Associates Publishers.

Medina-Mora, R, Wong, H and Flores, P 1992. The ActionWorkflow Approach to Workflow Management. Paper Presented at *4th Conference on Computer-Supported Cooperative Work (CSCW'92)*, Toronto, Canada.

Mohan, C 1997. Recent Trends in Workflow Management Products, Standards and Research. Paper Presented at *Proceedings NATO Advanced Institute (ASI) Workshop on Workflow Management Systems and Interoperability*, Istanbul, Turkey.

Muth, P, Weissenfels, J and Weikum, G 1998. What Workflow Technology can do for Electronic Commerce. Paper Presented at *Euro-Med Net '98 Conference, Electronic Commerce Track*, Nicosia, Cyprus.

Nardi, B 1995. Some Reflections on Scenarios. In *Scenario-based Design - Envisioning Work and Technology in System Development* (Carroll, J), 387-399.

Nielsen, J 1995. Scenarios in Discount Usability Engineering. In *Scenario-based Design - Envisioning Work and Technology in System Development* (Carroll, J), 59-84.

Niessink, F and Vliet 1998. Two Case Studies in Measuring Software Maintenance Effort. Paper Presented at *International Conference on Software Maintenance Information Systems*, Bethesda, Maryland.

Nissen, HW and Jarke, M 1999. Repository Support for Multi-Perspective Requirements Engineering. *Information Systems - Special Issue on Meta Modeling and Method Engineering* 24(2): 131-158.

Nuseibeh, B, Kramer, J and Finkelstein, A 1994. A Framework for Expressing the Relationships Between Multiple View in Requirements Specification. *IEEE Transactions on Software Engineering* 20(10): 760-773.

OMG 1999. *Unified Modeling Language Specification*. Version 1.3, http://www.omg.org.

Parnas, D 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 15(12): 1053-1058.

Pohl, K 1994. The Three Dimensions of Requirements Engineering: A Framework and its Applications. *Information Systems* 19(3): 243-258.

Potts, C 1994. Software Engineering Research Revisited. *IEEE Software* 19-28.

Potts, C and Hsia, I 1997. Abstraction and context in requirements engineering: Toward a synthesis. *Annuals of Software Engineering* 3 23-61.

Potts, C and Newstetter, WC 1997. Naturalistic Inquiry and Requirements Engineering: Reconciling Their Theoretical Foundations. Paper Presented at *International Conference on Requirements Engineering (ICRE'97)*, Washington, DC.

Potts, C, Takahashi, K and Anton, AI 1994. Inquiry-Based Requirements Analysis. *IEEE Software* 11(1): 21-32.

Ramage, M and Bennett, K 1998. Maintaining Maintainability. Paper Presented at *International Conference on Software Maintenance*, Washington.

Regnell, B 1999. *Requirements Engineering with Use Cases - a Basis for Software Development.* PhD Thesis, Department of Communication Systems, Lund University, Lund Institute of Technology.

Regnell, B, Kimbler, K and Wesslén, A 1995. Improving the Use Case Driven Approach to Requirements Engineering. Paper Presented at *Second IEEE International Symposium on Requirements Engineering (ICRE '95)*, York, UK.

Rogers, Y and Ellis, J 1994. Distributed Cognition: An alternative framework for analysing and explaining collaborative working. *Journal of Information Technology* 9 119-128.

Rolland, C and Achour, B 1998. Guiding the construction of textual use case specifications. *Data & Knowledge Engineering* 25(1): 125-160.

Rolland, C, Achour, CB, Cauvet, C, Ralyté, J, Sutcliffe, A, Maiden, N, Jarke, M, Haumer, P, Pohl, K, Dubois, E and Heymans, P 1998a. A proposal for a scenario classification framework. *Requirements Engineering Journal* 3(1).

Rolland, C, Souveyet, C and Achour, CB 1998b. Guiding Goal Modeling Using Scenarios. *IEEE Transactions on Software Engineering* 24(12): 1005-1071.

Schäl, T 1996. Workflow Management Systems for Process Organisations. Berlin: Springer-Verlag.

Schmidt, K 1991. Riding a Tiger, or Computer Supported Cooperative Work. Paper Presented at *2nd European Conference on Computer-Supported Cooperative Work*, Amsterdam, The Netherlands.

Schmidt, K 1994. Social Mechanisms of Interaction. *COMIC, Esprit Basic Research Project 6225.*

Schmidt, K and Bannon, L 1992. Taking CSCW Seriously: Supporting Articulation Work. *Computer Supported Cooperative Work. An International Journal* 1(1-2): 7-40.

Schmidt, K and Simone, C 1996. Coordination Mechanisms: Towards the conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work* 5 155-200.

Sheth, A and Kochut, KJ 1997. Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Colloboration Systems. *Advances in Workflow Management Systems and Interoperability.*

Sommerville, I 1992. Software Engineering. Wokingham: Addison-Wesley.

Star, S 1995. The Politics of Formal Representations: Wizards, Gurus, and Organizational Complexity. In *Ecologies of Knowledge. Work and Politics in Science and Technology* (Star, S), 88-118.

Strassl, J 1996. Reasoning about work activity during CSCW systems design. Paper Presented at *11th Meeting of CSCW-North Special Interest Group*, Durham, UK.

Strassl, J and Smith, SR 1997. Modelling and reasoning about work activity during systems design for cooperative working. Paper Presented at *Workshop on Representations in Interactive Software Development*, London, UK.

Strassl, J and Smith, SR 1998. Using scenarios to determine and represent requirements for workflow systems. Paper Accepted at *Workshop of Understanding Work and Designing Artefacts*, York, UK.

Strauss, A 1986. Work and the Division of Labor. *The Sociological Quarterly* 26(1): 1-19.

Suchman, L 1987. Plans and Situated Actions. The problem of human-machine communication. Cambridge, UK: Cambridge University Press.

Suchman, L 1994. Do Categories Have Politics? The language/action perspective reconsidered. *Computer Supported Cooperative Work* 2(3): 177-190.

Symon, G, Long, K and Ellis, J 1996. The Coordination of Work Activities: Cooperation and Conflict in a Hospital Context. *Computer Supported Cooperative Work* 5 1-31.

Weidenhaupt, K, Pohl, K, Jarke, M and Haumer, P 1998. Scenario Usage in System Development: A Report on Current Practice. *IEEE Software* 33-45.

WfMC 1998. Workflow and Internet: Catalysts for Radical Change, *Workflow Management Coalition*, http://www.wfmc.org.

Winograd, T 1994. Categories, Disciplines, and Social Coordination. *Computer Supported Cooperative Work* 2(3): 191-197.

Winograd, T and Flores, R 1986. Understanding Computers and Cognition: A new foundation for design. MA: Addison-Wesley.

Yourdon, E 1989. Modern Structured Analysis. Englewood Cliffs, NJ: Yourdon Press.

Yu, E and Mylopoulos, J 1994. Understanding "Why" in Software Process Modelling, Analysis, and Design. Paper Presented at *Proceedings of 16th International Conference on Software Engineering (ICSE '94)*, Sorrento, Italy.

Yu, ES 1995. *Modelling Strategic Relationships for Process Reengineering*. PhD Thesis, Graduate Department of Computer Science, University of Toronto.

Zave, P 1997. Classification Of Research Efforts In Requirements Engineering. *ACM Computing Surveys* 29(4): 315-321.